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	February 2002 (M 11)
Actual date of delivery	6.3.2002
Deliverable number	D9
Deliverable name	Report on modelling of domain-specific knowledge
Type	Report
Status & version	Final
Number of pages	19
WP contributing to the deliverable	WP 5.1
WP / Task responsible	NISLab, University of Southern Denmark
Other contributors	Frank Steffens, Bosch, André Berton, DaimlerChrysler
Author(s)	Niels Ole Bernsen, Marcela Charfuelan, Laila Dybkjær, Mykola Kolodnytsky
EC Project Officer	Domenico Perrotta
Keywords	VICO domain knowledge, VICO dialogue design, route task, help task

Formateret

Abstract (for dissemination)	This report from the IST/HLT VICO (Virtual Intelligent Co-driver) project describes the current state of work and future plans concerning domain modelling, task modelling, and dialogue design for the modelled domains and tasks. The tasks addressed in the report are route navigation and help information on the VICO system.

Table of contents

Report on modelling of domain-specific knowledge

Niels Ole Bernsen, Marcela Charfuelan, Laila Dybkjær, Mykola Kolodnytsky

1. Introduction

VICO aims to demonstrate in-car multilingual natural spoken interaction including a modest amount of multimodal interaction for a number of tasks. The current VICO task list is as follows, divided into *committed tasks* which will certainly be implemented and *optional tasks* some of which will be implemented:

Committed tasks

1. simple route task (address queries only)
2. help task
3. petrol station task (may be a POI task)
4. hotel reservation task
   4.1. hotel identification
   4.2. hotel reservation
   4.3. go to hotel (a POI task)
5. car manual task
6. point of interest (POI) tasks
7. car device operation task
8. customised sightseeing tour

Optional tasks

~~8.~~9. complex route task (finding locations and planning routes using, e.g., "near" or "via")

~~9.~~10. location task (the driver asks for the current position)

~~10.~~11. news reading task

Formateret: Punktopstilling

A *domain* is a topical area, such as road navigation or news services, within which an interactive system allows users to accomplish one or several tasks. A *task* is a goal-directed user activity, such as aiming to obtain specific kinds of information or making a device do certain things. VICO is a *task-oriented* spoken language dialogue system (SLDS) which will enable users to accomplish multiple tasks through spontaneous dialogue. This implies that, conceptually, the task has a certain priority over the domain, the implication being that we only analyse the domain to the extent that domain knowledge is required for task specification. As the above VICO task list shows, the VICO tasks address a number of different domains, i.e. road navigation (to addresses, petrol stations and other POIs), system (VICO) information, hotels, car manuals, car devices, and news services. Altogether, this is a considerable number of domains and tasks within those domains, and the domains and tasks in question have very different characteristics. Moreover, indications are that the tasks to be handled by VICO are not entirely independent from each other.

This requires us to consider how to flexibly combine tasks when designing the driver's dialogue with VICO.

Given the way we work in the VICO project, we largely proceed by specifying and implementing one task at a time. We might have done things otherwise. In particular, we might have specified all committed VICO tasks before writing a single line of code, spending the first 12-18 months of the project on task analysis and specification. However, as we learn a great deal from actually implementing a task – about general system architecture, module communication, system management, properties of modules we do not implement ourselves, properties of ready-made resources and devices which will be used in the project, patterns of within-project collaboration, etc. – it would seem good SLDS engineering practice to collect construction and implementation experience in parallel with task analysis and task-oriented dialogue specification. Also, having a task implemented in running software enables us to conduct increasingly realistic user experimentation with the system, enabling us to iteratively improve the software and its usability as well as gathering much-needed linguistic data. Moreover, as VICO is a relatively ambitious research endeavour [Steffens et al. 2001], we cannot take much for granted as we proceed. This means that, at the time of writing, we have focused our domain and task analysis efforts on a subset of the tasks mentioned above.

It needs to be added, though, that we only *largely* proceed by specifying and implementing one task at a time. Given the domain and task diversity in VICO, domains and tasks impose very different requirements to VICO's high-level software architecture, including the software architecture of VICO's NLU *(natural language understanding)*, DM *(dialogue manager)*, and RG *(response generation)* modules. To minimise the risk of having to substantially re-implement those modules, it is necessary to look ahead to those not yet fully analysed tasks in order to identify important structural properties which affect, or might affect, the high-level NLU/DM/RG architecture. At the moment, we have identified two such structural properties. The first one is the *user modelling module.* This module will enable VICO to adapt to different drivers according to their particular characteristics. It is necessary early on to analyse the complexity of incorporating a user modelling module in VICO in order to prevent later discoveries to force substantial re-implementation. The second structural property is the already mentioned property of *task interdependence.* The VICO tasks are not entirely independent of one another. For instance, the driver may legitimately begin to negotiate one task with VICO, interrupt the negotiation in order to solve another task, and then revert to the negotiation of the first-mentioned task. For these reasons, we are presently analysing and specifying the user modelling functionality of VICO, and we have developed a draft task grammar which will handle VICO task interdependencies. The work on the task grammar, in its turn, has required a certain amount of preliminary analysis of the VICO tasks which are not planned to be incorporated in the first prototype.

More specifically, the first of the two planned VICO prototypes (due autumn 2002) will solve the following tasks:

1. simple route task
2. help task
3. hotel reservation task
    a. hotel identification
    b. hotel reservation

c. go to hotel (a POI task)

4. point of interest (POI) task (to an, as yet unknown, extent).

In addition, the first VICO prototype will incorporate a 0.1 version of the VICO user model.

Among those tasks, the simple route task has been analysed, specified and implemented in running software. The VICO help task has been specified and implementation will commence around 1 March 2002. Specification of the hotel task, which actually consists of three sub-tasks (cf. the list above), will commence on 1 March 2002 when a final list of hotel properties has been agreed with VICO partner ITC-Irst. After that, we will address the POI task.

The present report, therefore, describes work done on domain and task analysis for the simple route task and the VICO help task. Figure 1 shows the VICO dialogue manager architecture.

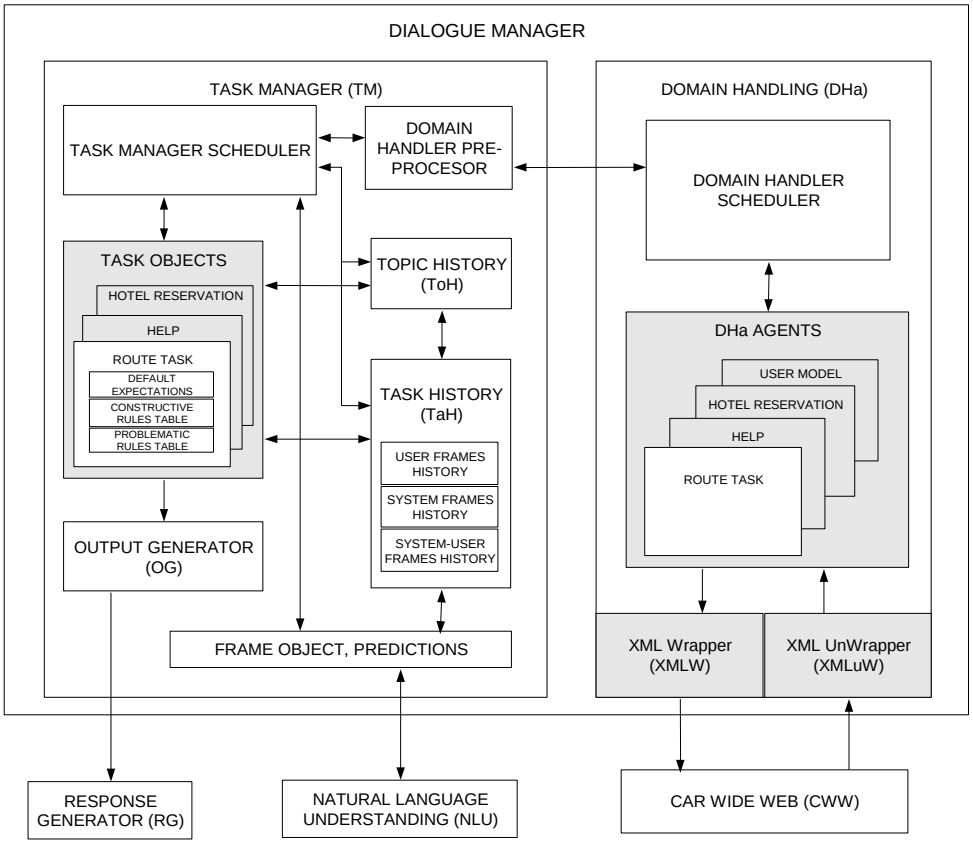

Figure 1. VICO dialogue manager architecture.

2. Methodology for domain and task analysis

Domain and task analysis for interactive systems design is a complex scientific subject in its own right. Domains and tasks are extremely different and leave dialogue designers very different degrees of freedom in the way they design the dialogue for a specific task. Very roughly, it is necessary to analyse the relevant objective properties of the domain, the objective requirements for accomplishing the task, the intended user population, users' conceptions and misconceptions of the domain and the task, users' abilities wrt. accomplishing the task using various candidate modalities and devices, the environment in which the users interact with the system, as well as several other aspects, and always on the backdrop of a series of high-level constraints on system development, such as the resources available [Bernsen et al. 1998, Geutner et al. 2001].

Having done those analyses, one can proceed to designing the actual dialogue which will enable users to accomplish their tasks. The dialogue design should respect every important detail discovered in the domain and task analysis process, in order to design a usable solution. The design process proceeds at ever-increasing levels of detail until it becomes possible to program directly from the design specification.

Analysis of the simple route task, for instance, primarily concerns what is involved in negotiating with the driver which address the driver wants to go to. Fortunately, virtually all drivers can be assumed to know what an elementary address is, i.e. that an elementary address involves items such as city names, street names, street numbers and the like. So, on the face of it, designing for the simple route task is a rather objective endeavour. The VICO help task, on the other hand, poses very different problems. The notion of "help the user wrt. VICO" is an extremely vague one: help with what, exactly? At which level of detail? What do users want help with? Etc. The VICO help task domain is a *semi-open* one, i.e. it is not possible to objectively circumscribe what users might want help with. To satisfy the users, semi-open domains either require a human expert at hand, and we don't have a homunculus available in the car, or a relatively strong-handed dialogue design which makes clear to users up front what they can get help with, and how.

In the following, we describe our current solutions to those two very different tasks, the simple route task and the VICO help task.

3. The simple route task

The simple route task may be contrasted with the complex route task. The *simple route task* consists in making VICO understand to which address one wants to go. VICO's back-end system will then provide the driver with the necessary driving instructions. By contrast, *the complex route task* provides the driver with a number of additional constraints to impose, such as specifying *intersections*, asking to go to the goal *via* some specified location or to go someplace *near* someplace else, etc. The following sub-sections discuss the main problems which have been addressed or, in some cases, remain to be fully addressed, in order to analyse and design the simple route task.

3.1 Input combinatorics

In the absence (at the time) of overviews of the commercial databases from TeleAtlas for Greater London, Germany, and the Italian Trentino province, we have conceptualised the simple route task as involving five hierarchically ordered parameters or *items: part of country, city, part of city*

(which may be absent for small cities), *street*, and *street number* (which may be absent in the database for some streets). Relative to some item, an item higher up in the hierarchy is called a *superordinate* item and an item lower in the hierarchy is called a *subordinate* item. This yields a combinatorics of 31 simple route input possibilities from the user, including improbable input such as a city and a street number. The point here not only is that the user may input any item combination in fully unconstrained speech. More importantly, perhaps, is the fact that even if the user inputs less improbable item combinations, VICO may not recognise or understand all of it. In addition, the user may of course produce null input, or the user may input ambiguous input, such as a proper name which happens to match both a street name and a city name in the NLU lexicon. Finally, the user may produce a type error, such as calling what is actually a city database entry a street. In such cases, the NLU must report to the DM that the NLU only knows, e.g., "Munich" as a city but that the user claims that "Munich" is a street. In total, users may produce 34 different kinds of address input.

It is important to note that this model assumes that the NLU lexicon is able to *type* the proper names produced by the driver as being cities, streets, etc. It would seem far preferable for this proper name typing to take place in the NLU rather than having the DM take care of it.

As we assume distinction between three confidence score levels characterising the NLU's input to the DM (high, medium, low), we get a total of 102 different possible *first-time* user inputs. When VICO responds to first-time user input, the user may continue the dialogue by adding information to what was produced in the first input, revising the information VICO believes to have received, asking VICO to repeat, etc. In other words, the input space for the simple route task is a rather large one, since route task completion, given a certain amount of user corrections and VICO-initiated meta-communication, may involve up to, at least, ten turns by the user and ten turns by VICO.

3.2 Input processing

To process the simple route task input, VICO's DM makes use of a number of separate modules, including the *Task Manager* (TM) which receives input from the NLU and decides what to do with it, the *Domain Handler* (DHa) which helps the TM by querying the domain database via the Car-WideWeb (CWW) whenever the TM needs it, the *Task History* (TaHi) which keeps a record of the progress in solving particular tasks, the *Topic History* (ToHi) which keeps a full and structured record of the interaction with the user, and the *User Model* (UM) which informs the TM about relevant user properties to which VICO should adapt.

The user's input may not be qualified for handling by the DHa. In such cases, the TM handles the input itself by producing output to the user via the DM's Output Generator (OG). These cases include lowest-confidence score input, null input, ambiguous input, type errors, and input which does not include address items but includes relevant other information, such as confirmation, negation, repeat, etc.

If the user's input is qualified for being handled by the DHa, the DHa queries the database and returns to the TM output belonging to one in several different *cases*. Thus, the queried items may be *unique and incomplete* when there is only one database hit but the database includes more items than were in the query, *up to three* when there are two or three database hits, *more than three* when there are more than three database hits, *complete* when the database has no more items than provided in the query, *inconsistent* when no address exists complying with the queried items. In addition, the DHa informs the TM whether or not a particular street *has street numbers*

included in the database. This is done in order to be able to inform drivers who ask for a particular street number that numbers are not available for that street if this happens to be the case, and to be able to offer drivers the opportunity to specify street numbers when these are available in the database.

To do its job, the DHa is able to perform a modest amount of reasoning. For instance, to resolve inconsistency, the DHa performs *constraint relaxation* in order to be able to confirm and return to the user at least a single correct address item from which to start further negotiation. Moreover, if the TM has forwarded to the DHa only items *subordinate* to city, the DHa assumes that the query is local and does not query the entire database. Other case-specific rules are applied if, for instance, the query includes part of country and street number only, in which case the street number is being ignored. For instance, a driver who asks for, or, more likely, is wrongly being understood by VICO ask asking for, number 12 in Bavaria, would not really be helped if VICO in response provides a list of all streets in Bavaria which are known to the database to include number 12.

3.3 Output processing

The TaHi keeps track of the progress in the negotiation with the driver. In particular, if the driver initially provides incomplete address input, such as a city without a street, the driver is asked once if the driver wants to provide additional information on the task, such as a street name and number, before VICO proceeds to plan the route.

Based on the case returned from the DHa and information from the TaHi, the TM instructs the OG about the information to send to the RG. The TM's instructions are formal (conceptual) expressions containing the relevant items. Based on those expressions, the OG specifies sentence referents (numbers) and specifies the received items according to its grammar (including word order, repetitions, agreement relations, etc.). The specification is sent to the RG which retrieves phrases and sentences from its database, and performs the necessary item substitutions, transforming the received information into a text string which is then played by the *Speech Synthesiser* (SS).

The transformations performed sequentially in the chain TM -> OG -> RG may be simple *phrase mappings*, as in the confidence score = 1 (lowest) case: route_confirm, go_where? sent by the TM, mapped into <6, 5> by the OG, and converted into "You want navigation assistance. Please name the part of country or a city." by the RG through a simple table look-up. At the other end of the complexity continuum, there are complex *phrase, item, and type mappings* involving *grammar application* performed by the OG, followed by *phrase and sentence retrieval and type and item substitution* performed by the RG. For instance, if the confidence score is 3 (highest) and the DHa case is that items are non-unique (2 or 3 items to select from), the TM outputs: several [items] /optional/ superordinate [item + superordinate], [item + superordinate], /optional/ [item + superordinate], choose [item + superordinate]. The OG transforms this expression into: <15: [type: plural] [item] [optional: list 2 or 3 superordinate [superordinate items], 16>. And the RG inserts the received types and items into the looked-up sentence templates: "There are several [insert type: plural] named [insert item] //optional: in [insert superordinate item]//. These are [insert item] in [insert superordinate item], //optional: [insert item] in [insert superordinate item]//, and [insert item] in [insert superordinate item]. Which one of these destinations would you like to go to?"

In addition, the RG sends a brief version of its item output to the car display as: Goal: [items]. This solution was chosen on the basis of Wizard of Oz experiments which compared three display output solutions: (a) one in which only the final negotiated goal appeared on the display, (b) one in which the goals being negotiated were briefly presented on the display, and (c) one in which all VICO spoken output appeared on the display [Bernsen and Dybkjær 2001].

3.4 Handling complexity

As shown above, the route task spoken I/O space is potentially quite large and complex, as it is a function of the user's unconstrained spontaneous initiative in the route task dialogue, item combinatorics, confidence score levels, recogniser errors, NLU errors, user input ambiguity, user input type errors, domain complexity, such as the 21 or so cities named Neustadt in Germany, the dialogue strategy of offering the driver a second opportunity to provide additional item information, and the error handling strategies employed by VICO. This complexity is handled by two complementary dialogue strategies, the *constructive input strategy* and the *difficult input strategy*.

3.4.1 Strategy for handling constructive input

In what we call the constructive input strategy, the user continues to provide the input needed to complete the route task as quickly and efficiently as possible. The dialogue proceeds in two main stages which are kept track of by VICO. In the first stage, called *first-time user input*, VICO's task is to (i) identify the navigation task, obtain, and get user confirmation of, at least one unique address item. Once this has been done, VICO proceeds to (ii) the *second-time user input* stage at which its task is to obtain any additional address items the user wants to commit to. Following that stage, VICO can tell the user that the route is being planned and let the system's navigation back-end take over. Constructive input strategy dialogues may of course involve misrecognitions, ambiguity, inconsistency, etc., and, in particularly efficient dialogues, the two stages just described may become a single stage. This happens when the user successfully provides a unique and complete set of items the first time around. However, any constructive input strategy dialogue may at any point lapse into a difficult input strategy dialogue as discussed in the next section.

3.4.2 Strategy for handling difficult input

Whereas the terminal node in a constructive input strategy dialogue is VICO's output, unopposed by the driver: *The route is being planned*, the terminal node of difficult input strategy dialogues is VICO's output: *I am sorry that I cannot understand what you are saying. You may want to let VICO close down and then try again.* During difficult input strategy dialogues, VICO's key objective is to avoid that terminal node. VICO does that by systematically "stepping backwards" using *graceful degradation* in the dialogue any time it receives problematic input, such as an empty frame from the NLU, non-constructive "No" replies from the driver, lowest-confidence level input, etc. Stepping backwards means simplifying the dialogue with the driver through a series of steps, including, among others: *Where would you like to go? Please name the part of country or a city.; Do you want help with navigation or information about VICO?; Please ~~spell~~ say navigation if you want navigation help or VICO if you want VICO information.; Is there somewhere else you would like to go?, Please spell the part of country or the city you want to go to.;* etc. The goal of VICO's stepping backwards consistently is to get the dialogue back on track by identifying at least a single driver objective which the driver can confirm. Once this has been

achieved, the dialogue moves forward instead of backwards, and starts employing the constructive input strategy. Only if all fails is ~~is~~ the driver told that VICO is unable to continue the dialogue with the driver.

3.5 Towards general meta-communication in VICO

General VICO meta-communication means a general strategy for handling a number of basic problems of communication, such as repeated lowest-level confidence scores. In this context, "general" means that the strategy will be applied in a task-independent fashion. It is too early to develop such a general strategy at this stage. The reasons are two-fold: (i) we need to have several different tasks implemented in order to investigate what may work across tasks, and (ii) we need to work with an integrated speech recogniser of the right kind, i.e. one which accepts spontaneous unconstrained user input in one or several of the VICO languages. The recogniser which is integrated at the moment works for US English and does not accept unconstrained spoken input. It makes little sense to commit to general basic error handling strategies as long as we do not know the particular behaviour of the VICO recogniser. On the other hand, the multiple-task nature of VICO offers an excellent testbed for investigating the extent to which general error-handling strategies are meaningful in advances SLDSs. So, we are committed to pursuing this line of investigation which could yield important insights for developing much-needed general meta-communication functionality.

3.6 Mapping the simple route task into the TeleAtlas databases

Until this point, we have worked with the route task using an in-house, hand-crafted address database for the purpose. To ensure subjects' familiarity with its contents and hence optimise realism in their use of the database, the database includes Danish locations. Moreover, the database enables users to put together any possible address query case (cf. Section 3.2). We are in the process of analysing the four TeleAtlas databases for Denmark, Germany, UK/London, and Trentino, respectively, in order to specify solutions for how to map the existing route task dialogue structure onto those databases. When the mapping principles have been identified, we will move to using the TeleAtlas databases, providing users much more freedom in building realistic route queries.

4. The VICO help task

4.1. Introduction

What follows is a first specification of the VICO help task, i.e. the task in which VICO provides information about itself. Given the semi-open nature of the help task which makes help tasks notoriously difficult to design well, it is not feasible to give users unconstrained initiative in addressing the task (cf. Section 2). Instead, this first version of the help task is specified in a *system-directed* fashion, providing a limited number of options for the driver. Experimentation will show if this is satisfactory to users or whether drivers need more help options.

In addition, a first model is proposed for the repeat task (Section 4.6).

Several times in the text below, it says "to be confirmed" or "to be completed". This is because we still need confirmation (or an alternative strategy, or to agree on certain parameters) from VICO partners on points relevant to the VICO help task output contents. Once this has happened, this document will be revised (or completed). However, it would seem that implementation is

ready to begin because those possible revisions will only affect the contents of output expressions and not the structure of the Help Module.

Also, it is remarked several times that VICO help contents will, or may, have to be revised when we add new tasks or new functionality to VICO. Again, this would not seem to prevent us from implementing the Help Module now. The modularity of the system ensures that output can be re-designed independently of the basic structure of the DM.

4.2. Task structure analysis

The task structure could be the following: a general entry to VICO help information which provides access to a three-item menu. The user should be able to access the sub-menu items directly, which is why we need several frame slots for the task.

1. VICO help (main menu).
   1.1. What can VICO do?
   1.2. How to operate VICO.
   1.3. VICO technology.

So, the task has a main menu item which subsumes three sub-items.

4.2.1. VICO help

This is the main entry to VICO help. It basically invites the user to choose between the three sub-entries.

4.2.2. What can VICO do?

4.2.2.1 *Negotiate where to go to the following goals:*

- locations specified by part of country, city, part of city, street, and street number (when available);
- hotels specified by name.

4.2.2.2 *Negotiate which hotel to contact by telephone using the following parameters:* [TO BE CONFIRMED]

- [to be completed]

4.2.2.3 *Provide help information about VICO.*

Note that this information will have to be updated each time we add new tasks. The program should be prepared for that.

4.2.3. How to operate VICO

4.2.3.1 *When to speak to VICO*

Speak to VICO after pushing the button. A green light on the screen shows when VICO is listening. If the light turns red VICO is no longer listening and you should press the button again before you speak to VICO. Note that this solution to the question of how to inform the driver that VICO is listening is still being compared with other possible solutions.

4.2.3.2 *How to speak to VICO*

Speak quite normally to VICO but remember that VICO only understands information about the topics it can handle. VICO prefers that you stick to the point.

Note that this information may have to be updated if and when we add to or change the VICO interface.

4.2.4. VICO technology

This is the longest part of the VICO information output. It is mostly there for fun, i.e. for use by those drivers who take a technical interest in the system, rather than for providing operational information about VICO. The proposed VICO output is in the dialogue structure specification below.

Note that this information may have to be updated when we add functionality to VICO.

4.3. Dialogue structure specification

In the following output text specification, a paragraph break represents the insertion of a pause in the output.

4.3.1. VICO help

The driver says, e.g., *I would like to have information about VICO, VICO, tell me about yourself,* or simply *Help*.

This produces the VICO help frame and ticks off its help slot.

If NLU confidence level (CF) = 3, RG sends the following string to SS:

You have asked for information about VICO. Please say if you want information about what VICO can do, how to operate VICO, or how the VICO technology works. Just select one of these topics and then select another topic later, if you want. You may also want the information repeated.

If CF = 2, RG sends to SS:

Do you want information about VICO, please say yes or no.

If CF = 1, RG sends to SS:

Do you want help with navigation or information about VICO?

Note that this information (when CF = 1) will have to be updated each time we add new tasks. The program should be prepared for that.

4.3.2 What can VICO do?

The driver says, e.g., *Tell me what VICO can do.*

This produces the VICO help frame and ticks off its domain slot.

If CF = 3, RG sends the following string to SS:

VICO can help you with the following tasks:

First, to get to the following destinations: addresses specified by some or all of the following: part of country, city, part of city, street, and street number. [TO BE CONFIRMED]

Second, to ~~get phone numbers for making~~select a hotel and make a hotel reservation~~s~~. To find your favourite hotel, you may specify your wishes for the following: [TO BE CONFIRMED] [to be completed]

Third, VICO can explain what VICO can do, how to operate VICO, and how the VICO technology works.

You may want this information repeated, ask how to operate VICO, ask how the VICO technology works, or tell VICO where you want to go.

Note that this information will have to be updated each time we add new tasks. The program should be prepared for that.

If CF = 2, RG sends to SS:

Do you want information about what VICO can do, please say yes or no.

If CF = 1, RG sends to SS:

Do you want help with navigation or information about VICO?

Note that this information (when CF = 1) will have to be updated each time we add new tasks. The program should be prepared for that.

4.3.3. How to operate VICO

The driver says, e.g., *Tell me how to operate VICO.*

This produces the VICO help frame and ticks off its operation slot.

If CF = 3, RG sends the following string to SS:

Speak to VICO after pushing the button. A green light on the screen shows when VICO is listening. If the light turns red VICO is no longer listening and you should press the button again before you speak to VICO.

Speak quite normally to VICO but remember that VICO only understands information about the topics it can handle. VICO prefers that you stick to the point.

You may want this information repeated, ask what VICO can do, ask how the VICO technology works, or tell VICO where you want to go.

Note that this information will have to be updated each time we add new tasks. The program should be prepared for that.

If CF = 2, RG sends to SS:

Do you want information about how to operate VICO, please say yes or no.

If CF = 1, RG sends to SS:

Do you want help with navigation or information about VICO?

Note that this information (when CF = 1) will have to be updated each time we add new tasks. The program should be prepared for that.

4.3.4. How does the VICO technology work

The driver says, e.g., *Tell me about the VICO technology.*

This produces the VICO help frame and ticks off its technology slot.

If CF = 3, RG sends the following string to SS:

VICO is an advanced prototype spoken dialogue system. The system is quite complex internally but the basics of how it works are rather simple. The main components which process your input and make the system's output audible and visible are the speech recogniser, the natural language understanding module, the dialogue manager, the response generator, the speech synthesiser, and the screen. To support these components in doing their jobs, the GPS, or global positioning system, keeps track of where your car is at the moment, the databases contain the information

needed for navigation and hotel reservation, the car wide web passes GPS and database information to the modules, and the system manager keeps track of all communication within the system.

Note that this information may have to be updated when we add new functionality to the system. The program should be prepared for that.

When you have pushed the button to speak, VICO goes active. VICO's speech recogniser is prepared to listen and convert your spontaneous spoken utterance into a text string of words.

The text string is then passed to VICO's natural language understanding component for analysis. The result of the analysis is a distillation of the meaning of what you said which is passed to the dialogue manager, the real brain of VICO.

The dialogue manager analyses what you said in the context of the dialogue so far. Depending on the result of the analysis, VICO may, for instance, ask ~~its domain handler~~the database manager to query the geographical database or consult the global positioning system. VICO then specifies the response it has decided to give and sends the specification to the response generator.

The response generator determines the actual wording to be used in the response. This wording is sent to the speech synthesiser which produces VICO's spoken response.

In addition, VICO presents the most important information on the screen.

If your goal is to be directed to a destination, and once you have agreed with VICO where to go, VICO will give you driving instructions as appropriate without the need for further dialogue. The reason VICO can do that is that VICO constantly consults its global positioning system to know the position and direction of your car.

Please note that today's speech recognition technology is not perfect. VICO may sometimes fail to ~~hear~~ understand you correctly. In such cases, VICO will try to get the dialogue back on track by using its dialogue repair strategies.

You may want this information repeated, ask what VICO can do, ask how to operate VICO, or tell VICO where you want to go.

Note that this information will have to be updated each time we add new tasks. The program should be prepared for that.

If CF = 2, RG sends to SS:

Do you want information about how the VICO technology works, please say yes or no.

If CF = 1, RG sends to SS:

Do you want help with navigation or information about VICO?

Note that this information (when CF = 1) will have to be updated each time we add new tasks. The program should be prepared for that.

4.4. Frames

We seem to need two new frames for the VICO information task, one of which can be used more generally:

3.1. The VICO Help frame including the following slots: help, domain, operation, technology.

3.2. Repeat.

Both frames will go to the TM. We do not seem to need an agent, like the domain handler, for dealing with VICO help information because no inferencing and no CWW database lookup would seem to be involved. So, TM will have to deal with the frames, inserting them into the dialogue history (or histories), passing them on to the VICO Help Module, and, based on that module's output to TM, producing a semantic instruction for OG to pass on to RG. The instruction depends on the CF score level (cf. above).

4.5. Additional help input

Obviously, the driver may continue the dialogue with VICO on the help task beyond what has been envisioned above where only the user's first help input is considered. ~~Proposed~~ General strategies for handling the help and repeat tasks are proposed in Tables 1 and 2 below. Their implementation will give us some experience with general error correction strategies, i.e. strategies which can be generalised across the board. We will have to further develop those strategies based on much more data and more tasks than we have at present.

Table 1 shows the help task dialogue structure.

Driver 1	C Score	VICO 1	Driver 2	Driver 2	VICO 2	Driver 2	VICO 2
Information	CS=3	Do/Op/Tech?	jump	no CS 1-3	S1	probl CS 1-3	Nav. or help?
	CS=2	Help yes/no?	loop/**end**	yes/no CS 1	Nav. or info.?	probl CS 1-3	Nav. or help?
	CS=1	Nav./help?	**end**/loop	no CS 1-3	S1	probl CS 1-3	Nav. or help?
Domain	CS=3	Domain expl.	**end**/jump	-	-	probl CS 1-3	Nav. or help?
	CS=2	Dom. yes/no?	loop/**end**	yes/no CS 1	Nav. or info.?	probl CS 1-3	Nav. or help?
	CS=1	Nav./help?	**end**/jump	no CS 1-3	S1	probl CS 1-3	Nav. or help?
Operation	CS=3	Ops. expl.	**end**/jump	-	-	probl CS 1-3	Nav. or help?
	CS=2	Ops. yes/no?	loop/**end**	yes/no CS 1	Nav. or info.?	probl CS 1-3	Nav. or help?
	CS=1	Nav./help?	**end**/jump	no CS 1-3	S1	probl CS 1-3	Nav. or help?
Technology	CS=3	Tech. expl.	**end**/jump	-	-	probl CS 1-3	Nav. or help?
	CS=2	Tech. yes/no?	loop/**end**	yes/no CS 1	Nav. or info.?	probl CS 1-3	Nav. or help?
	CS=1	Nav./help?	**end**/jump	no CS 1-3	S1	probl CS 1-3	Nav. or help?

Table 1. Help task dialogue model.

Explanations

Driver x = driver's input (numbered).

VICO x = VICO's output (numbered).

Jump = driver addresses a different help sub-task.

Loop = driver addresses the same help (sub-)task.

End = driver leaves the help task (ends or addresses a different (non-help) task). End is in boldface in Table 1 because, for the help task designer, it is good news when the user leaves the help task :-)

End = driver either wants to leave the help task as a whole or at least the subordinate help sub-task VICO asked about (see comments below).

Boldface = end nodes.

S = output sentence.

S1 = How may I help you?

The three Driver 2 columns (4, 5 and 7 from the left) are alternatives (three different groups of cases). In Column 4, the driver responds appropriately with CS = 3 or 2 and loops, jumps or ends, as the case may be. "/" splits the driver's dialogue moves in correspondence with the alternatives offered by VICO in Column 3.

In Column 5, the driver responds as stated and with the CS stated. Column 6 shows VICO's answer to the respective utterances (with their CSs)in Column 5. Column 7 includes the really problematic driver responses to VICO 1, such as empty or unintelligibility frames. Column 8 shows VICO's responses to column 7 driver responses.

This dialogue model may do for now. However, there are at least two unsolved problems:

1. The driver who persistently loops with CS = 1. I suggest that we leave this problem aside for the moment because it seems to be something which can happen in any task. Therefore, it seems preferable to wait and, at some later stage when we have more tasks implemented, do a focused test of these cases and try to come up with a single general solution. We also need to see how the recogniser behaves.

2. The driver who just says "no" with CS = 2 or 3 within one of the three subordinate sub-tasks (domain, operation, technology). These cases are marked by **end** in Table 1. Given that the driver just says "no", it is not clear that this driver actually wants to end the information task as a whole. The driver might just want a different subordinate sub-task. I suggest that VICO replies with S1 to these (not very helpful) drivers.

4.6. Repeat

Repeat is not addressed in Table 1 because it needs a separate frame. See Table 2.

Driver 2	C Score	VICO 2	Driver 3	VICO 3	Driver 3	VICO 3	Driver 4	VICO 4
Repeat	CS 3	Repeats	-					
	CS 2	S1	yes CS 2-3	Repeats	no CS 2-3	Continue		
	CS 1	S1	yes/no CS 1	S2			yes/no CS 1	S3

Table 2. Repeat task dialogue model.

Explanations

Driver x = driver's input.

VICO x = VICO's output.

Repeats = VICO repeats its latest output.

S = output sentence.

S1 = *Do you want me to repeat? Please say yes or no.*

S2 = *Say again.*

S3 = *How may I help you?*

Continue = VICO continues the output from where it stopped when the driver seemed to ask for repeat.

The two column pairs Driver 3- VICO 3 represent alternative cases.

Let us consider this repeat task dialogue model a general and experimental one to be used in all cases where the user asks VICO to repeat something. S3 may not be the best solution to repeated problems with yes/no answers from the driver. However, this is a more general issue since we will have yes/no questions all over the place. Once we have implemented some more, let us do a focused study of how VICO handles yes/no questions.

5. Testing and evaluation

NISLab's system test setup was changed in late November 2001 from a Wizard of Oz setup based on NISLab's Magic Lounge system into a car simulation environment. Magic Lounge is a virtual multi-party meeting environment in which more than two people can conduct spoken dialogue and exchange text messages over medium-bandwidth networks. In the Magic Lounge environment, the text facility emulated VICO's text display. All communication was logged, videotaped and transcribed. In the present setup, a 42" screen acts as the car's wind screen, the driver has a steering wheel and pedals used to control driving in the computer car game *(Need for Speed)* we are using, and an adjacent portable provides the VICO text display facility. Its name notwithstanding, the car game actually can be configured in a way which makes all test subjects feel at ease driving the car. Excluding the expensive screen and the portable PC which we had already, the price of this setup is about one per cent of the cheapest car simulator on the market. One drawback is that we are not able to log the driver's actual driving behaviour but have to obtain this data through on-location observation (contrast the Bosch car simulator described in Manstetten et al. 2002). All communication is logged, recorded and videotaped. All dialogues are transcribed. We are presently preparing to analyse the data gathered during two sets of experiments made with this setup in December 2001 and January 2002. In the next set of experiments to be done in March 2002, we will add the help and hotel reservation tasks to the scenarios to be solved by our subjects.

6. References

Bernsen, N. O., Dybkjær, H. and Dybkjær, L.: *Designing Interactive Speech Systems. From First Ideas to User Testing.* Springer Verlag 1998.

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

Geutner, P. et al.: Report on market situation, technological trends and user expectations. IST/HLT VICO Deliverable D3, June 2001.

Manstetten, D. et al.: Evaluation report from simulated environment experiments. IST/HLT VICO Deliverable D7, January 2002.

Steffens, F. et al.: System Specification. VICO Deliverable D4, June 2001.