Specification Techniques for Multi-Modal Dialogues in the U-Wish Project

Anton Nijholt, Betsy van Dijk and Olaf Donk
Centre of Telematics and Information Technology (CTIT)
University of Twente, PO Box 217
7500 AE Enschede, the Netherlands
anijholt@cs.utwente.nl

ABSTRACT

In this paper we describe the development of a specification technique for specifying interactive web-based services. We wanted to design a language that can be a means of communication between designers and developers of interactive services, that makes it easier to develop web-based services fitted to the users and that shortens the pathway from design to implementation. The language, still under development, is based on process algebra and can be connected to the results of task analysis. We have been working on the automatic generation of executable prototypes out of the specifications. In this way the specification language can establish a connection between users, design and implementation. A first version of this language is available as well as prototype tools for executing the specifications. Ideas will be given as to how to make the connection between specifications and task analysis.

Keywords: specification, web-based services, multi-modal dialogues, virtual worlds.

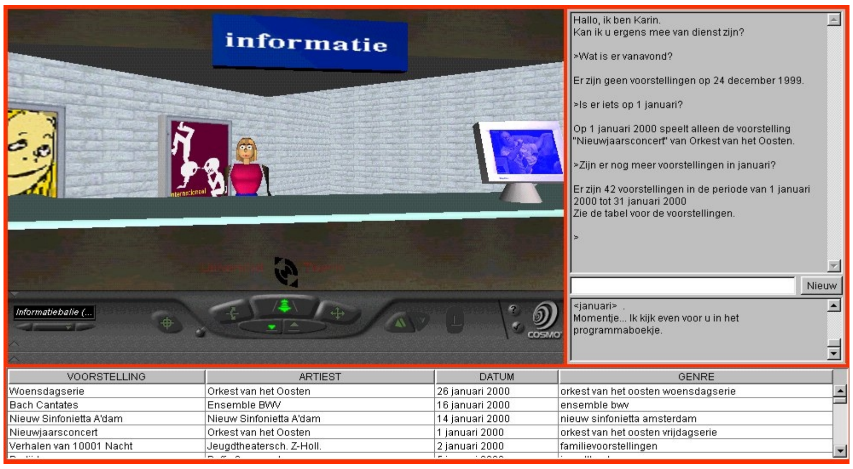
1. INTRODUCTION

We give an overview of our research done on developing a specification technique that is not only suited for describing interactive web-based services and the interactions between users and the systems, but that can also serve as a means of communication between designers and developers of such systems. A language that is suited for this use has not been developed before. We are planning to base it on existing speciication languages as much as possible. The actual use of the specification language under development is illustrated by the specification of aspects of an information and transaction system embedded in a virtual environment. For those interested, the complete specifications of the system can be found in [4]. The development of this system is part of the U-WISH project.

The U-WISH (Usability of Web-based Information Services for Hypermedia) project [6] aims at a user-centred design method for web-based services that guides the iterative process of software development. This is a Dutch project coordinated by the Dutch Telematics Institute. In the first phase of the project guidelines, techniques and tools for the specification and assessment of web-based services are being developed. In the next phase, these usability-engineering elements will be integrated into a coherent design method for web-interfaces and subsequently applied. Our institute's part in this large project focuses on specific aspects of interaction in Web-based information and transaction environments, i.e. the effects of different interaction styles, and possible schemes for allocating and integrating modalities in multi-modal dialogues. Until now much effort has been given to the defintion of a language that makes it possible to describe and discuss multi-modal interactions.

First we will discuss this topic in order to describe our field of research in more detail. One approach is to define modality as a communication channel. In this way speech, gestures, typed language and so on are different modalities. In systems using different modalities, a multi-modal dialogue may be created by using these different channels simultaneously. Such an approach is used to create a multi-modal environment in the DIVE- system [2], developed at SICS. In this case, a multi-modal dialogue is created by resolving anaphores in the natural language used by taking into account mouse-clicks.

In some systems different windows on the screen are also considered different modalities. VMC [7] and ALFRESCO [10] are examples of systems using these notions. In the latter system ambiguities in the natural language used are resolved by using a mouse in a graphical 2D-environment. The VMC environment uses different windows to present information to the user. One of these windows displays a virtual world in which visitors can walk to an information desk. Behind the desk is an information and transaction agent called Karin (see Fig. 1) who can talk to the visitors using natural language interpretation and speech synthesis. The other windows display the dialogue that the user has



Hallo, ik ben Karin.
Kan ik u ergens mee van dienst zijn?

>Wat is er vanavond?

Er zijn geen voorstellingen op 24 december 1999.

>Is er iets op 1 januari?

Op 1 januari 2000 speelt alleen de voorstelling "Nieuwjaarsconcert" van Orkest van het Oosten.

>Zijn er nog meer voorstellingen in januari?

Er zijn 42 voorstellingen in de periode van 1 januari 2000 tot 31 januari 2000
Zie de tabel voor de voorstellingen.

>

<januari> .
Momentje... Ik kijk even voor u in het programmaboekje.

VOORSTELLING	ARTIEST	DATUM	GENRE
Woensdagserie	Orkest van het Oosten	26 januari 2000	orkest van het oosten woensdagserie
Bach Cantates	Ensemble BWV	16 januari 2000	ensemble bwv
Nieuw Sinfonietta A'dam	Nieuw Sinfonietta A'dam	14 januari 2000	nieuw sinfonietta amsterdam
Nieuwjaarsconcert	Orkest van het Oosten	1 januari 2000	orkest van het oosten vrijdagserie
Verhalen van 10001 Nacht	Jeugdtheatersch. Z-Holl.	2 januari 2000	familievoorstellingen

Fig. 1 The information desk with dialogue windows

with Karin and there is a table showing additional information about performances. On Karin's desk there is also a monitor showing previews of performances.

In our development of a specification language capable of describing multi-modal dialogues we will consider both different communication channels and different windows as being different modalities. The systems under study in the U-wish project – a company website, a web-accessible information system providing information about regulations on house property, premises, buying, etc., and the Virtual Music Centre (VMC) - focus on the use of natural language and graphical interfaces in combination with hypertext and virtual environments. We do not want to limit ourselves to these combinations when the expressive power of our specification language is concerned. We will only consider the specification of the VMC since it is the most advanced of the three systems and it is a multi-modal environment where we have the possibility of a natural fusion of the different modalities.

2. CHOICE OF SPECIFICATION TECHNIQUE

We will describe a number of criteria on which to score known specification techniques. These criteria also serve as guidelines in the development of the new specification language.

- **Ease-of-use:** As we want to create a useful specification language, we want this language to be easy to use. Difficult interaction processes that involve lots of actions may still be difficult to write down. But it should be possible to use our language to 'sketch' interactive systems.

- **Readability:** Specifications should serve as a medium of communication. The people that use them, therefore should be able to read them. As the specifications in the U-wish project should facilitate communication between web-designers and web-programmers special care should be taken on this point.

- **Modularity:** The use of modular specifications enables reuse of parts of the specification in other systems. If these modules can be structured in an hierarchical architecture, it is possible to specify both the details of a systems as well as the more abstract general behaviour.

- **Possibility of execution:** If tools exist or can be made for executing specifications, it may be possible to use the specifications for prototyping.

- **Identification of tasks:** If it is possible to identify tasks and actions in the specification explicitly, it may be possible to develop specifications based on task-models. Especially in the context of our U-wish project this is an interesting property.

- **Expressive power:** Some properties are more easily described in a certain specification language and other properties more easily in another language. It is important for the usability of a specification language that one is using the proper language, i.e. it should be possible to write the important characteristics down straightforwardly.

- **Possibilities for further development:** It is unlikely that a specification language exists that is 100 % suitable for our use. Therefore we are looking for a language that can be used as a basis for further research and that can be expanded. In order to be able to expand a language we must known at least its syntax and semantics.

3. KNOWN SPECIFICATION TECHNIQUES

In the following section we'll give a short description of a number of specification languages. We'll introduce: natural language, GOMS, GTA, HDDL, Z and CSP. For more thorough descriptions the reader is referred to the references given in the corresponding subsections.

Natural Language (NL): First of all, NL is not a formal method in itself. Its use lacks the precise and unambiguous communication that formal languages provide. The advantage of NL is its enormous expressive and adaptive power. It is used often together with formal languages in order to explain the formal descriptions as formal languages on their own tend to be difficult to understand. Furthermore NL is used to clarify the meaning of formal specifications.

GOMS: The GOMS technique (Goals, Operators, Methods, and Selection Rules) [3] is well suited for describing interactions. The GOMS method generates an analysis of the goals and sub-goals someone wants to achieve. The goals are ordered in a tree-like hierarchy. The operators in the GOMS description describe the actions the user of a system can perform. The methods are appropriate combinations of goals and operators, well-known to the user. Finally the selection rules describe how the user chooses between the various available methods at any time. A GOMS specification, however, doesn't describe the interacting systems. The technique describes tasks with respect to systems and organisations. It is possible to use GOMS to generate descriptions of human performance in advance based on specifications of the system the operator interacts with.

GTA: The GTA method (Groupware Task Analysis) [12] describes how a community of people works. The roles that various members of the community fulfill are described as well as how people cooperate. The actual work that is done is described by hierarchical task-description. The objects that are relevant and the relation these objects have to one another is described in an object-structure. The resulting specification takes also that part of the system the user deals with into account, i.e. the user virtual machine (UVM). The UVM determines the semantics of the atomic tasks. These work- and object-structure serve in the GTA method as a basis for ETAG descriptions of the task-analysis. ETAG (Extended Task Action Grammar) [11] is a declarative language in which tasks are described by decomposing them into hierarchical structured atomic actions.

HDDL: The HDDL language is developed by Philips for describing dialogues in automatic inquiry systems, involving speech recognition. It is a modular language, dialogues are decomposed into sub-dialogues and it incorporates a satisfying strategy for handling dialogue flow management. This strategy is explained in [1]. In real applications HDDL has proven to be a powerful language. In co-operation with a company that is changing its voice-response based market orientation to a telephone-based spoken dialogue systems orientation, we have contributed to the development of a HDDL telephone-based system for movie information and reservation, a similar system for information about governmental regulations – called PO Box 51 - and a HDDL based system for obtaining snow information in European ski-resorts.

Z: The Z language is a well known formal specification technique based on logics. Specifications in Z are well suited to described structures, objects and relations between them. Also constraints hereupon can be described in Z. The language is based on logics and mathematics and uses the corresponding notation style. It is possible to specify dynamic aspects of systems using Z, but these do not combine with Z naturally.

CSP: CSP (Communicating Sequential Processes) developed by Hoare is a kind of process algebra. In CSP concurrent processes and their interaction can be described. CSP is well suited to describe multiple simultaneously active processes. It is also possible to assert certain properties of the systems specified. E.g., it is possible to check whether the interactive system cannot deadlock.

4. CAPACITIES OF KNOWN TECHNIQUES

We will now discuss the advantages and drawbacks. The criteria discussed above serve as guideline.

- **Natural Language:** NL is not a formal method, but due to its adaptive capabilities it is useful for describing various systems. It is easy to use, but is also highly ambiguous and it lacks expressive power for precise descriptions. NL does not have the merits of formal languages. It is not possible to derive properties of systems described solely in NL and to generate executable prototypes out of NL specifications. We will use NL however for a first sketch of systems.

- **GOMS and GTA**: These are formal descriptions of (groups of) people and the tasks they perform. These descriptions can range from that of the use of a simple calculator to the description of the daily work a team of employees is doing. The identification of tasks of the user of a system can be easily identified. Both GTA and GOMS do not describe the systems people are working with. They describe the UVM at most, i.e. the systems as it appears to the user but not what is inside the system. These descriptions can not be made executable as the systems that should execute are not specified. Maybe it is possible to add this to the specifications, but then again a language is needed to expand these specifications.

- **HDDL:** Specifications of systems in HDDL are modular as the language demands a modular structure. HDDL is not really a description language, it is a programming language for spoken dialogue systems. It is not easy to identify the various tasks a user of the system performs. HDDL is tailored to handle questions, answers and other spoken utterances using a telephone, but not mouse-clicks, navigation in virtual worlds, use of agents and so on. Furthermore, it can not easily be adapted to future needs.

- **Z:** The Z-language is suited to make modular specifications with different levels of abstraction. A drawback of Z for our purpose is its notation. The abundant use of mathematical symbols makes Z difficult to read and a large variety of symbols is needed. Actually one does not need all the symbols as a lot of them are shortcut notations. Reducing the set of symbols will not make the specifying easier. A drawback of Z is the static nature of what is described. It is difficult to describe changing environments and interaction between system and users. This makes it difficult to generate executable code out of specifications.

- **CSP:** In contrast to Z, CSP is based on actions and interactive processes. From a CSP-specification it is therefore quite simple to identify the task the user performs. As process algebra is based on actions its use may also enable us to make the connection between the results of hierarchical task analysis and the specifications of actual systems. It is possible to group parts of a CSP specification into modules and hide details to the outside. In this way a CSP specification may be read at different levels of abstraction. A number of executable languages are developed based on CSP and process algebra, for example LOTOS [5]. The possibilities to make tools for executing seems promising and we have made prototypes of tools for this purpose that are still under development. We have decided to develop a specification language based on CSP. In the next section we will discuss this in more detail.

5. NEW SPECIFCIATION LANGUAGE

In a typical interactive system a number of processes are running at the same time and the user may choose to communicate with some of them. Generally these processes can be described by state transition diagrams. These diagrams are suitable to describe the possible actions of a process at a certain time as well as the communication a process is capable of. Although most processes in interactive systems can be described by state transition systems, the use of these diagrams is limited. For example it is not possible to create new instances of objects in a system that is totally described by state-based diagrams. In an actual application a number of processes is running simultaneously and the synchronisation of these processes is an important aspect when considering the exact behaviour of the system.

We have chosen to develop a specification language based on process algebra. This gives good insight in all possible actions at any moment and multi-modal dialogues can be handled. A process can handle input and output on different channels by being ready to perform different actions. Multimodal communication by using multiple windows can be described by running multiple processes simultaneously. The process algebra we use is a somewhat altered version of Communicating Sequential Processes (CSP). We have made some minor syntactic modifications to the language. For a more thorough description of the syntaxis and semantics of the CSP language we use, see [9].

As process algebra does not put any claims on the actual results of the actions that take place, the execution of specifications results in a trace of these actions, but not in any behaviour as such. We wanted to investigate the apparent behaviour of the system we have specified. Therefore we wanted to connect the specification and the actual appearance of the system. The execution of the specification would then result in a running model of the system. The reasons to do this are threefold. This connection will give us insight into the performance and limitations of the use of our language. It is easier to check the correspondence between the specifications we have made and the actual systems. Finally, the approach will enable us to use our language for prototyping purposes.

The demands on such a connection are severe, the mapping between the user interface (UI) and the processes of the specification should be a retrieve function $\rho\colon UI \to CSP$ [13]. It should be total and surjective. That is, there is a CSP state corresponding to each state of the UI, and also that each possible CSP state is possible in the UI. If insufficient care is taken in creating this connection the behaviour of the created interface can be different from the specification.

In order to establish this connection we include several directives in the specification. These directives are preceding each process declaration for which there is a corresponding process in the UI. In the specifications we are working on the directives communicate with a TCL/TK process that controls the interface. TCL is an interactive scripting language that can cooperate with the GUI toolkit, TK [8]. In the variant of the language to be developed later, we do not necessarily have to use it. It is not evident that all important characteristics of a user interface that we want to express can be programmed in TCL/TK. We believe however the connection as such is useful in order to increase the usability of our specification language. The behaviour of the execution of the interface may serve as a dynamical documentation of the specification and hence increase the readability of the specification. In [9] we describe a specification of the VMC in which we have used these TCL/TK directives and we give an example of the possible looks of the resulting executable specifications.

6. SPECIFICATION OF THE VMC SYSTEM

We describe how we have developed the specifications of the VMC system. The complete specifications can be found in [4]. Of course the specifications do not contain all details of the system specified. We have included those aspects that make up the general outline. Those parts of the system that actually interact with the user or are meant to help the user to orientate within the system are specified in more detail. We have not specified multiple instances of similar objects.

6.1 Natural Language

We start with a NL description to sketch properties and behaviour of the systems. It serves as a starting point for the final, formal specification. The formal specifications are not final. Insight gained will serve later as guideline to further develop our language and refine the specifications.

VMC is a virtual environment that contains a number of interactive components. On the walls are posters that serve as hyperlinks to information about theater performances. If necessary a browser is started when a poster is clicked. Doors can be opened and closed again, the user may make music on a (virtual) synthesizer keyboard and so on. A special functionality is embodied by an information board displaying a map of the theater. This map is a teleport and clicking it will transport the user to the corresponding place in the building. Inside the VMC there is an information desk. The application that handles the desk is called THIS (THeater Information System). In the current implementation it is also accessible when the user is not near the information desk. There is interaction between the VMC and the THIS system. The user may refer to system utterances that are presented in a table with clickable items (information about performances). See again Fig. 1.

THIS is an application that communicates with the user using NL. The user can ask about specific performances or about the scheduled performances during a certain period of time. THIS will answer by using NL (both speech synthesis and written in a frame on the web page) or by filling in a table. The user may ask new questions by clicking in the table or by typing new questions. Whenever the user wants to make a reservation the system will ask for the necessary information, like number of persons and possible discount, that is not available to the system from the previous conversation. Finally there is a button to restart the system.

6.2 Formal Specifications

We will describe the CSP specification made of the VMC system. The natural language descriptions given above will serve as a backbone. The descriptions given will be in a simple CSP variant. First we will specify the apparent behaviour of THIS when the user doesn't use the mouse.

THIS: The user can only enter questions by entering them in the entry field. Also Karin, the embodied agent of the system, cannot answer by using the table (Fig. 1). The system consists of five parallel processes, one for each frame, one for the user and one representing Karin.

```
This where This = Entry || Dialogue || Karin || User || Status
```

The user can only enter utterances. Of course the user can watch the systems behaviour, but for obvious reasons this part of the user is not modelled. In a more recent research version of the system we experiment with an eye-tracking system to let the system know which part of the web page the user is looking at. Clearly, then the user's gaze behavior becomes part of the input modalities of the system and it has to become part of the specifications.

```
User = (entryfield -> User)
```

The entry field accepts an utterance, passes it to the dialogue recorder and clears itself. Finally it passes the utterance to Karin. This process is repeated over and over.

```
Entry = (entryfield ->
        entrytodialogue ->
        clearentryfield ->
        passtokarin ->
        Entry)
```

The dialogue recorder accepts alternating utterances from the entry-field and Karin. It starts with waiting for an utterance, as Karin is starting the conversation. The status-process just goes on and on accepting status messages. The actual displaying is not modelled, it is assumed that the displaying is a side-effect from the acceptance of a message.

```
Dialogue = (karintodialogue ->
            entrytodialogue ->
            Dialogue),
Status = (passtostatus -> Status)
```

Karin consists of two parallel processes, Karinstart and Iq. Iq models the conversation of Karin. It accepts an utterance and reacts by returning a number of status messages and an answer to the question.

```
Karin = Karinstart || Iq,
Iq = (passtoiq -> Iqreact),
Iqreact =  [(getfromiq -> Iq)
           [] (getstatus -> Iqreact)]
```

The process Karinstart starts a new conversation and makes Karin welcome the user. Once the conversation is started Karin accepts questions, passes them to her IQ and returns the reactions from Iq to the dialogue recorder.

```
Karinstart = (passtoiq ->
                   getfromiq ->
                   karintodialogue ->
                   Karinbehaviour),
Karinbehaviour = (passtokarin ->
                   passtoiq ->
                   Waitreaction),
Waitreaction =  [ (getfromiq ->
                   karintodialogue ->
                   Karinbehaviour)
               [](getstatus ->
                   passtostatus ->
                   Waitreaction)]
```

Specification of the complete dialogue: Processes Table, Mouse and Nieuw_button are added to THIS resulting in:

```
This = Entry
```

```
            || Dialogue
            || Status
            || Table
            || Karin
            || User
            || Mouse
            || Nieuw_button
```

As it is possible to ask questions by clicking in the table the process Dialogue is changed into:

```
Dialogue = (karintodialogue ->
                [ (entrytodialogue ->
                    Dialogue)
                [](questiontodialogue ->
                    Dialogue)])
```

The process Mouse doesn't care much, it is an abstract version of the actual program that interprets the mouse and manages the pointer and cursor. The action mouse_out takes place as the mouse is clicked outside the pop-up-menu containing the question about the selected performance.

```
Mouse = [ (nieuw_button -> Mouse)
          [](click_table -> Mouse)
          [](mouse_out -> Mouse)
          [](ask_table -> Mouse)]
```

The process Emptytable manages the behaviour of the table before Karin has written information into it. In the current THIS it is possible to click an empty performance. The process Filledtable speaks to itself. The process Select_menu models the behaviour of the table when a performance (possibly an empty one) is selected and the pop-up-menu is visible. The process Select models how the table behaves when the pop-up-menu has disappeared.

```
Table = Emptytable,
Emptytable = [ (fill_table -> Filledtable)
               [](new_table -> Emptytable)
               [](click_table -> Select_menu)],
Filledtable = [ (fill_table -> Filledtable)
               [](new_table -> Emptytable)
               [](click_table -> Select_menu )],
Select_menu = [ (mouse_out -> Select)
               [](ask_table -> question_out -> Select)],
Select = [ (fill_table -> Filledtable)
           [](new_table -> Emptytable)
           [](click_table -> Select_menu)]
```

The process associated with the button that starts a new dialogue, named 'Nieuw' in Dutch, simply waits untill it is clicked, then tries to reset Karin and restarts itself.

```
Nieuw_button = (nieuw_button ->
                resetkarin ->
                Nieuw_button)
```

Karin is much the same, she now can be reset and she accepts questions from the pop-up-menu. She also can answer by filling the table. This will always be accompanied by a verbal response in the dialogue window. The Iq process is slightly altered to allow Karin to be reset.

```
Karinbehaviour = [ (resetkarin -> new_table -> Karinstart)
                 [](passtokarin ->
                     passtoiq ->
                     Waitreaction)
                 [](question_out ->
                     questiontodialogue ->
                     passtoiq ->
                     Waitreaction)],
Waitreaction = [ (getfromiq ->
                 [ (karintodialogue ->
                     Karinbehaviour)
                 [](fill_table ->
                     karintodialogue ->
                     Karinbehaviour)])
                 [](getstatus ->
                     passtostatus ->
                     Waitreaction)],
Iq = [  (passtoiq -> Iqreact)
     [] (startiq -> Iqreact)],
```

The VMC Specification: A subset of the interactive world of VMC is modelled. The world, Vmc, consists of a number of parallel processes. The This-process is the Karin-system described in the previous section. The This_manager is used to activate and deactivate the processes under This by means of the actions this_on and this_off. So:

```
Vmc = Mouse
      || Avatar
      || World
      || Bookshelf
      || Poster
      || Door
      || Map
      || Browser_manager
      || This_manager
      || This
```

The processes that model the behaviour of books and posters evoke the Browser_manager to start up a new browser. The door can be opened and closed again.

```
Bookshelf = (book_click -> evoke_browser_book -> Bookshelf),
Poster = (poster_click -> evoke_browser_poster -> Poster),
Browser_manager = [ (evoke_browser_book ->
                    Browser_manager)
                  [](evoke_browser_poster ->
                    Browser_manager)],
Door = (door_click ->
        open_door ->
        door_click ->
        close_door ->
        Door)
```

The Map-process, that serves as a teleport in the VMC, communicates with the Avatar, that on its turn can respond to the map and to move-commands from the user given by the mouse. The Avatar-process checks with the world to see whether a move asked for by the mouse is possible or not

```
Map = (map_click -> goto_map -> Map),
Avatar = [ (try_move -> check_world ->
           [ (possible_move -> goto_step -> Avatar)
           [](not_possible_move -> Avatar)])
           [](goto_map -> Avatar)],
World = (check_world ->
        [ (possible_move -> World)
        [](not_possible_move -> World)])
```

The mouse that is used for most communication between user and system is modelled like this:

```
Mouse =   [ (book_click -> Mouse)
          [](door_click -> Mouse)
          [](map_click -> Mouse)
          [](poster_click -> Mouse)
          [](try_move -> Mouse)
[](activate_this -> Mouse)
[](deactivate_this -> Mouse)
[](nieuw_button -> Mouse)
[](click_table -> Mouse)
[](mouse_out -> Mouse)
[](ask_table -> Mouse)]
```

7. EVALUATION AND FUTURE PLANS

We compare the language with the criteria mentioned before. Based on our own experiences we think the language is easy to use and makes it possible to sketch interactive processes. We have found that it is also quite easy to specify interactive systems consisting of a number of simultaneously running components. The same observation is true for readability. However, we ourselves are used to our language; it will be necessary to introduce others to our language in order to obtain a less subjective view. The results of the developed language on our following criteria can be evaluated more objectively. It seems likely that it is possible to structure the specifications very well. CSP has good capacities of structuring different levels of abstraction. We have plans for expanding the CSP language with facilities for creating modular specifications. We are also developing tools for automatically generating executable code based on CSP, possibly expanded with TCL/TK-code, for generating the interface.

8. CONCLUSIONS

Our goal during the development of a specification language for interactive services has been to establish a link between users, design and implementation. Therefore we have tried on the one hand to take the users as a starting point for our specification. This is done by connecting process algebra and task analysis. We expect that this connection will prove useful but we can not state this for certain on this moment. We think that the research that is done so far can be powerful for bringing users and interactive services together. As our specifications can be fitted onto the results of task analysis, the resulting interactive services are tailored to fit the users. Also the generated executable prototype provides a way to evaluate the design of the services in an early stage of the development.

9. REFERENCES

[1] H. Aust & M. Oerder. Dialogue Control in Automatic Inquiry Systems, Proc. 9th *Twente Workshop on Language Technology*, T. Andernach et al. (eds.), 1995, 45-49.

[2] C. Carlsson, & O. Hagsand. DIVE - a Multi-User Virtual Reality System, *IEEE Virtual Reality Annual International Symposium (VRAIS)*, 1993, 394-400.

[3] S.K. Card, T.P. Moran & A. Newell. *The Psychology of Human-Computer Interaction*. Hillsdale, NJ, Lawrence Erlbaum Associates, 1983.

[4] O. Donk, B. van Dijk & A. Nijholt. U-WISH: Specification techniques for multi-modal dialogues. CTIT TR 99-13, University of Twente, 1999.

[5] Information processing systems -- Open Systems Interconnection -- LOTOS -- A formal description technique based on the temporal ordering of observational behaviour, ISO 8807:1989, 1989.

[6] M. Neerincx, J. Kerstholt, B. van Dijk, A. Nijholt, S. Pemberton & G. Spenkelink. U-WISH -- Web-based Services for Information and Commerce: User-Centered Design Principles, Methods and Applications. http://www.cwi.nl/projects/uwish/, 1998.

[7] A. Nijholt & J. Hulstijn. Multimodal interactions with agents in virtual worlds. In *Future Directions for Intelligent Information Systems and Information Science*, N. Kasabov (ed.), Physica-Verlag: Studies in Fuziness and Soft Computing, 2000, to appear.

[8] J. Ousterhout. Tcl: An Embeddable Command Language. Proc. *USENIX Winter Conference*, 1990.

[9] B. van Schooten, O. Donk & J. Zwiers. Modelling Interaction in Virtual Environments using Process Algebra. Proc. 15th *Twente Workshop on Language Technology*, A. Nijholt et al. (eds.), 195-212, 1999.

[10] O. Stock et al. ALFRESCO: Enjoying The Combination of NLP and Hypermedia for Information Exploration. *Intelligent Multimodal Interfaces*. M.T. Maybury (ed.), AAAI Press/MIT Press, 1993.

[11] M.J. Tauber. ETAG: Extended Task Action Grammar. In: *Human-Computer Interaction -- INTERACT '90*, D. Diaper et al. (ed.), Elsevier Science Publishers B.V. (North-Holland), 1990.

[12] G. vd Veer, B. Lenting & B. Bergevoet. *Groupware Taks Analysis: Modeling Complexity*. 1995.

[13] J. Zwiers, J. Coenen & W. P. de Roever. A note on compositional refinement. In: Proceedings of the *5th Refinement Workshop*, 342-366, 1992.