# Knowledge Based Speech Interfacing For Handhelds

C.K.Yang, L.J.M.Rothkrantz

Data and Knowledge engineering
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
L.J.M.Rothkrantz@cs.tudelft.nl

**Abstract.** Speech technology is rapidly developing and has improved a lot over the last few years. Nevertheless, speech-enabled applications have not yet become mainstream software. Furthermore, there is a lack of proven design methods and methodologies specifically concerning speech applications. So far the application of speech technology has only been a limited success. This Paper describes a project done at CMG Trade Transport & Industry BV. It is called SWAMP and is an example of the application of speech technology in human-computer interaction. The reasoning model behind the speech interface is based on the Belief Desire Intention (BDI) model for rational agents. Other important tools that were used to build the speech user interface are the Microsoft Speech API 5 and CLIPS.

## 1 Introduction

Speech is the most common mode of communication between people. Although speech communication is not a perfect process, we are able to understand each other with a very high success rate. Research has shown that the use of speech enhances the quality of communication between humans, as reflected in shorter problem solving times and general user satisfaction [1]. Furthermore, speaking to humans subjectively seems to be a relatively effortless task [2]. The benefits mentioned above are some reasons that have moved researchers to study speech interaction systems between humans and computers.

In September 1999 CMG Trade, Transport & Industry BV started the Wireless Automotive Messaging (WAM) project. Its purpose was to develop new wireless services in the field of traffic and transport. The WAM application is based on the Client-Server model. The server is stationary while the client travels with the user in his car. Because the clients are mobile, communication is based on wireless techniques.

This paper discusses the SWAMP (Speech Interfacing in the Wireless Automotive Messaging Pilot) project started in October of the following year. The purpose of the SWAMP project was to analyse if a speech interface is better suited for the WAM pilot. Therefore the WAM client is extended with a speech interface: the SWAMP client. This offers a way for the driver to interact with the system while his hands and eyes remain free, ideal for car driving situations.

## 2 The SWAMP Client

Speech interaction between the user and the SWAMP application is based on dialogues. Generally, the user starts a speech interaction by indicating (via speech) what his desires are. The system then leads the user through a dialogue in which it tries to retrieve information regarding these desires. If eventually all the necessary information is collected, the application takes the appropriate actions to realise the user's desires. The general assumption behind the speech interface is that the user wants to accomplish something with his utterances, i.e. he has a certain goal in mind. The set of all services the SWAMP application has to offer is just a subset of all the goals the user can possibly have. Goals that don't correspond to a service, however are beyond the domain of the speech interface and are ignored. The speech interface is divided into 3 components.

1. The speech recognition or ASR component: Its function is to recognise the user's utterance and transform it into a format that can be processed.
2. The dialogue management component: Its function is to process the input from the speech recognition component to figure out what the user wanted to accomplish and take the appropriate actions to realise the user's wishes. This component is the main focus in this paper.
3. The speech synthesis or TTS component: Its function is to generate speech output to the user.

Figure 1 gives a graphical overview of how the speech interface is implemented. The main application is the original WAM client modified in such a way that it can communicate with the dialogue manager.

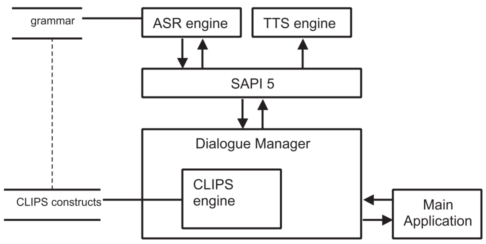


**Fig. 1.** Overview of SWAMP implementation

The SWAMP client is implemented in C++ (Microsoft Visual C++ 6.0 enterprise edition). Initially it was the intention to build the speech interface to run on Windows CE but due to limitations in software and hardware of handheld computers, Windows NT was ultimately chosen.

The Microsoft Speech Application Programming Interface 5.0 (SAPI5) is used as middle-ware between the engines and the SWAMP application. SAPI5 acts as a communication layer between the dialogue manager and the speech resources (ASR and TTS engine). It takes care of hardware specific issues such as audio device management and removes implementation details such as multi-threading. This reduces the amount of code overhead required for an application to use speech recognition and synthesis. Another advantage of using middleware is that the choice of the final ASR and TTS engine can be postponed till a later stadium (e.g. until there is more budget for better engines). The CLIPS expert system tool is designed to facilitate the development of software to model human knowledge or expertise. CLIPS is embedded in the SWAMP client. It can be viewed as the knowledge processing and management unit of the dialogue manager.

## 3 Dialogue Design

With each initial utterance from the user, the speech interface tries to find the corresponding service involved. Once the goals of the user are clear it tries to accomplish the service by checking whether all the information needed is available. If this is not the case, the speech interface must initiate a dialogue to retrieve the required information from the user until the task can be performed. All possible dialogues that the speech interface can be involved in must be designed beforehand. This includes speech prompts for each situation, and all possible user responses on those prompts. Furthermore design involves the definition of a grammar that captures the syntax of whole conversations into a few simple grammar rules.

### 3.1 Design Approach

The goal of the speech interface is to give a user access to the SWAMP services by means of simple speech interaction. To achieve this, one can choose between two different approaches: 1) demand a longer learning time for the speech interface and require the user to adapt his speaking style or 2) make it easy for the user by allowing an extensive grammar and modelling more and more complex dialogues so that the user can speak to the system as with another human.

Speech User Interface (SUI) designers have learned that humans are extraordinarily flexible in their speech and readily adapt to the speaking style of their conversational partners. This is not a new finding: think about how easily we adjust our speech depending on whether we are speaking to children or other adults. This flexibility has useful implications for designing the speech interface: after extensive use of the speech interface (as the user gets acquainted with the grammar and has more experience) some dialogues become less and less common. Since the user will adapt his style of interacting and refrain to only those dialogues that were successful in the past. Because of this finding and the choice of our typical user ("he is familiar with current computer technology") the first approach was chosen: only model the most common utterances and let the user adapt to it.

### 3.2 Dialogue Representation

Without a proper representation technique, the dialogues can quickly become very complex and unmanageable. In this project dialogues are represented by flow diagrams containing nodes representing start/begin points of a dialogue, boxes representing actions (e.g. an utterance from a user or an action from the system), diamonds representing decisions point and arcs to connect the nodes, boxes and diamonds. A dialogue always begins with a start node and ends with an end node. Within these nodes, the dialogue travels from box to box along the arcs and branching at the decision diamonds. A successful dialogue corresponds to a path in the flow diagram from the start node to the end node. Speech dialogues are context sensitive. In our representation, the context is defined by the positions within the dialogue flow. Each box represents a certain state or context. The arcs branching from a box indicate the options available within that context and the branches leading to a box define how that context can be achieved.

The power of above dialogue representation technique lies in the fact that dialogues are represented in a generic way. E.g. the (user action) boxes define what the user can say at that moment in the dialogue, but not how it must be said (this is defined in the grammar). In this way, a single path in the dialogue flow diagram can represent whole categories of similar dialogues.

A well-modelled dialogue flow diagram is one where each possible dialogue flow can fit in. This implicates that common communication errors, such as misunderstandings, should be modelled as well as mechanisms for correcting and preventing these errors, such as requests for confirmation and roll back. Table 1 shows an example dialogue for the kilometre registration (KM registration) service. The flow of this dialogue fits into the flow diagram in Fig. 2 (accentuated).

**Table 1.** Example dialog

| |
|---|
| U: Change trip type |
| S: Is it a business or a private trip? |
| U: It's a business trip? |
| S: OK, what's the project ID for this business trip? |
| U: Project ID is SWAMP |
| S: Do you want to set the project ID to SWAMP? |
| U: Yes |
| S: OK, trip type is set. |

In practice the dialogues can become so complex and the dialogue flow diagrams so large that it is best to split them up into one main dialogue and several smaller sub dialogues. For each sub dialogue a separate dialogue flow diagram is designed and referred to in the main dialogue flow diagram (by means of sub dialogue nodes). Another use for the dialogue flow diagrams occurs during the testing phase. Since each path from the start node to the end node corresponds to a successful dialogue. The correctness of the implementation of the dialogues can easily be verified if all the paths in the dialogue flow diagrams can be traversed

### 3.3 Grammar

The SAPI5 design specification requires the grammar of an application must be a context-free grammar (CFG) written in a format specified in the SAPI5 grammar schema. This schema describes the SAPI 5.0 speech recognition grammar format and is based on the XML framework. The ASR engine uses the CFG to constrain the words contained in the user's utterance that it will recognise. Basically the grammar file consists of a set of grammar rules in the grammar schema syntax. The complete specification of the schema can be found in the SAPI5 online help. Grammar rules can have an activation state, which can be set to active of inactive. SAPI5 recognises active rules and conversely does not recognise deactivated ones. The application may change the state of the rules during execution. So if a rule is no longer needed, it may be deactivated.

In order to indicate the functional parts of a sentence i.e. the parts that actually contain relevant information, the CFG can be extended with semantic information declared inside the grammar. This enables the ASR engine to associate certain. Recognised word strings with name/value-meaning representations. The dialogue manager then applies these meaning representation associations to understand and control the dialogue with the user.

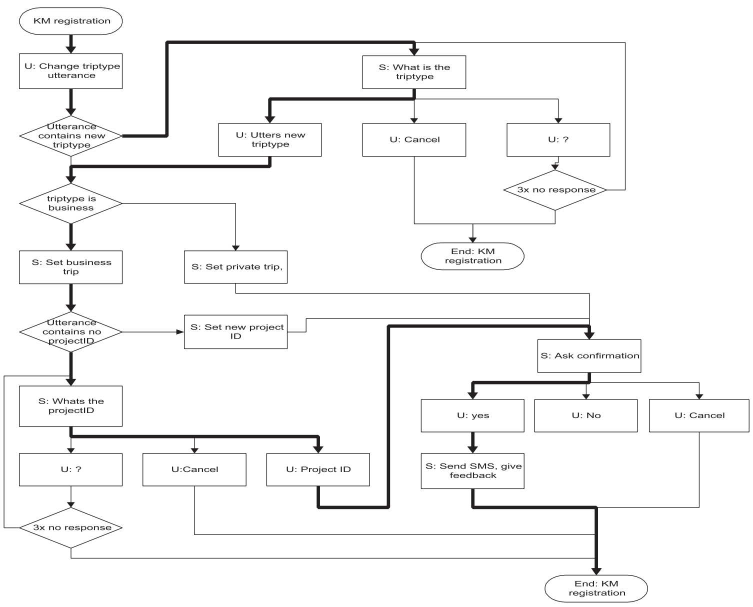


**Fig. 2.** Dialogue flow diagram for the KM registration service

The grammar rules are derived from a corpus of utterances by hand. Crucial in this process is the determination where the relevant information is located within an utterance. Once this is accomplished, the derivation process is straightforward.

## 4 The Reasoning Model

In the search for a suitable reasoning model for the dialogue manager: one that is capable of adequately describing the reasoning behaviour of the dialogue manager, the Belief-Desire-Intention (BDI) model [4] was chosen. In an implementation of a dialogue manager according to this model, the dialogue manager continuously executes a cycle of observing the world and updating its beliefs, deciding what intention to achieve next, determining a plan of some kind to achieve this intention, and then executing the plan.

There exist a correspondence between concrete CLIPS data structures and the attitudes in the BDI model. Beliefs in the BDI model are implemented as facts and rules in CLIPS. Facts are used to construct the dialogue manager's internal representation of the world. Facts can be seen as propositions and thus can only consist of a set of literals without disjunction or implication. Therefore special rules (belief rules) are used to complete the representation of beliefs. Belief rules represent the general relationship between facts (e.g. IF utterance=help THEN AlertLevel=high).

One way of modelling the behaviour of BDI reasoning [3] is with a branching tree structure, where each branch in the tree represents an alternative execution path. Each node in the structure represents a certain state of the world, and each transition a primitive action made by the system, a primitive event occurring in the environment or both. In this formal model, one can identify the desires of the system with particular paths through the tree structure. The above description of the branching tree structure is logically similar to the structure of the dialogue flow diagrams described in the previous section. In fact, both structures represent exactly the same: a path through the dialogue flow diagram is a successful dialogue, which is also a desire and therefore a path through the branching tree of the BDI reasoning model. As a result, the dialogue flows diagrams can be treated as the structures that describe the behaviour of the dialogue manager. They are directly implemented in CLIPS rules; each rule corresponds to a branch in the dialogue flow. Rules are both the means for achieving certain desires and the options available for the dialogue manager. Each rule has a body describing the primitive sub goals that have to be achieved for rule execution to be successful. The conditions under which a rule can be chosen as an option are specified by an invocation condition. The set of rules that make up a path through the dialogue flow, correspond to a desire.

The set of rules with satisfied invocation conditions at a time T (the set of instantiated rules) correspond to the intentions of the dialogue manager at time T. Obviously the intentions of the system are time dependent. The dialogue manager adopts a single-minded commitment strategy, which allows continuous changes to beliefs and drops its intentions accordingly. In other words the intentions of the system can be affected by the utterances of the user in contrast to blind commitment in which an intention is always executed no matter changes in beliefs.

## 5 An Example

In the previous section it was shown that the desires of the dialogue manager can be represented by dialogue flow diagrams. The flow diagrams are systematically translated into an executable system formulated in CLIPS rules. This section discusses the implementation of the desires. In particular the heuristics used for the translation from dialogue flow diagrams to CLIPS rules.

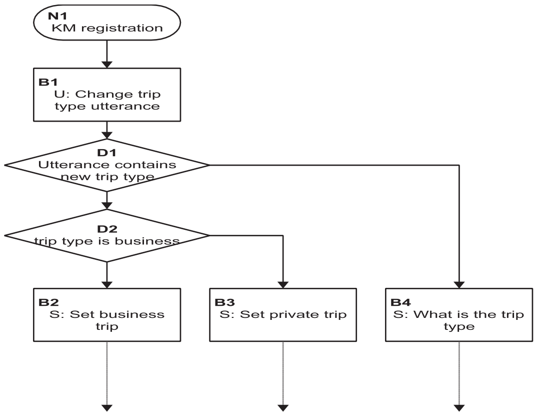


**Fig. 3.** Part of the dialogue flow diagram for KM registration service

Suppose we must transform a dialogue flow diagram as in Fig. 3. This dialogue is initialised when the user utters a phrase that matches the grammar for a change trip type utterance (box B1). Notice that box B1 has 3 branches (to the boxes B2, B3 and B4), furthermore we see that the action in B1 is a speech action from the user. From this we conclude that the dialogue flow should be implemented using 3 speech rules. The invocation conditions for each rule are the evaluated values of the expressions in the decision diamonds D1 and D2. The body of each rule contains the actions specified in the corresponding destination boxes. Furthermore, the body of the rules also contain actions to anticipate what follows after the action e.g. after box B4 the user must supply the new trip type so the grammar rules for trip type utterances should be activated.

The CLIPS rule in Fig. 4 corresponds to the branch from box B1 to B2 (Fig. 3). The keyword RECOGNISED in line 2 indicates that a user's utterance is recognised. The grammar rule that matched the utterance is $VID\_KMREG\_TRIPTYPE$.

```
1    (defrule KM_Registration_Business
2       ?in<-(RECOGNISED 161 VID_
              KMREG_TRIPTYPE 50 TripType
              business)
3       ?pos<-(POSITION MAIN RUNNING)
4       =>
5       (printout t "<SAY>Do you want set
         the triptype to business?</SAY>
6                    <ACT>VID_YESNO</ACT>
7                    <DEACT>"?*Mainrules*"</DEACT
9                    <REACT></REACT>" crlf)
10      (retract ?in)
11      (retract ?pos)
        (assert (POSITION MAIN KMREG))
        (assert (WANT CONFIRM))
        (assert (QUESTION KMREG business))
     )
```

**Fig. 4.** CLIPS rule - part of the KM registration service

Furthermore, the property name TripType with value business satisfies condition D1 and D2. The actions taken satisfy B2 (between the $< SAY >$ tags in line 5) and anticipate future utterances of the user by activating the $VID\_YES\_NO$ grammar rule and de-activating all other main grammar rules. Thereby limiting the user's input to only boolean values. The other actions in the rule body are used to update the internal representation of the world.

## 6 Conclusions

The model presented here allows for man-machine speech interaction. Indeed the speech interface of the SWAMP application implemented according to this model is capable of handling simple dialogues with the user. The dialogues are described and visualised as generic flow diagrams resembling branching tree structures [3]. The chosen representation technique has also contributed greatly to the containment of the complexity in the dialogue models. Furthermore it allows an easy translation to executable CLIPS rules.

## References


1. Chapanis, A. "Interactive Human Communication: Some lessons learned from laboratory experiments", In: Shackel, B. (eds). "Man-Computer Interaction: Human Factor Aspects of Computers and people", Rockville, MD: Sijthoff and Noordhoff, page. 65, 1981.
2. Nusbaum, H., et al., "Using Speech recognition systems: Issues in cognitive Engineering", In: Syrdal A. et al. (eds), Applied Speech Technology, Boca Raton, CRC press, page. 127, 1995.
3. Rao, A. and Georgeff, M., "BDI Agents: From Theory to Practice", in Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 1995
4. Wooldridge, M., "Reasoning about Rational Agents", The MIT Press, Cambridge, Massachusetts, 2000.