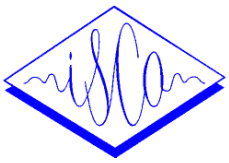

4th European Conference on
Speech Communication and Technology
EUROSPEECH '95
Madrid, Spain, September 18-21, 1995

Constraining of Input Media in a Spoken Dialogue System

Anders Baekgaard
ab@cpk.auc.dk
Center for PersonKommunikation (CPK)
Aalborg University, Fredrik Bajers Vej 7, DK-9220 Aalborg
Denmark

Abstract

A platform for spoken dialogue systems and a dialogue description formalism on which the platform is based is briefly described. The paper gives details about how input devices can automatically be constrained by the dialogue manager on the basis of the dialogue context.

The method is formalised and generalised and apply for all input media in a dialogue system, especially the speech recogniser and graphical input devices.

10.21437/Eurospeech.1995-295

1 Introduction

In this paper a platform for spoken dialogue systems is described with special focus on one aspect of a spoken dialogue system: the method used for constraining the speech recogniser and other input media.

First, a very brief description of the platform is given, then a dialogue description formalism (DDL - Dialogue Description Language) is presented in some detail, and finally the method for constraining input media (devices in our terminology) is described.

The platform (called GDS - Generic Dialogue System) consists of a number of modules that constitute the core functionality of a dialogue system together with a formalism for describing dialogues. The platform provides a design and development environment with support tools for interactive design and test of dialogues. The platform is originally developed within the ESPRIT II SUNSTAR project and used there for various speech controlled systems. Following this project, it has been further developed within the Danish Spoken Language Dialogue Systems project within the domain of flight ticket reservation and information. See [Baekgaard 1995] and [Dalsgaard and Baekgaard 1994] for descriptions and further references.

The architecture of the platform is designed to be modular and open such that it can be easily expanded. All modules of the dialogue system adhere to a well-defined protocol, and the architecture allows for multiple input and output media. Further, it interfaces to an APSG based NL component that is further described in [Music 1993].

2 Why constrain input devices?

The advantages of constraining the input devices in a real-time dialogue system are

- It improves the accuracy of the speech recogniser since the search space is smaller.
- It increases the potential maximum size of the language model because only parts of the language model need to be activated in a given dialogue state.
- It allows visual feed back to the user of possible input for graphical input devices as user actions that are not possible can be inactivated ("grayed out").
- It increases the processing performance of the speech recogniser.
- The constraints can be applied also on the NL parser to increase processing performance.
- It insures correspondence between possible input and the dialogue description thereby eliminating the need for error handling in the dialogue description.

The constraints are calculated on the basis of the dialogue model and the current dialogue context and constitute the set of input events that can be handled by the dialogue manager. Input events which are not within this set can not be handled by the dialogue manager - hence, constraining of input devices does not influence the dialogue. There is no idea in allowing e.g. the speech recogniser to recognise an utterance that the dialogue manager will not understand.

In the flight ticket reservation system there is approximately 600 word forms, but only at most 110 word forms are active in any dialogue state.

Constraining of a speech recogniser has been described by others, e.g. [Young 1990]. The method described here formalises and generalises the constraining mechanism to apply for all input devices.

3 The DDL dialogue description formalism

The dialogue description formalism is used by a dialogue designer to describe the dialogue between the human and the computer. It is defined as a formal visual language, and it is a recursive transition diagram-based language with compound visual and textual notations. Special features in the formalism are dedicated for spoken language dialogue handling. The formalism has several abstraction mechanisms for input and output which, at the highest level, allow the dialogue structure to be described in terms of semantic exchanges between the human and the computers. DDL is a compound language that consists of three layers: A graphical layer (GL) which is basically a graphical language for describing state transition diagrams for event-driven systems, a frame layer (FL) that provides form like specifications of variable length arrays of objects with associated actions, and a textual layer (TL) which is a full-featured textual programming language. DDL descriptions are interpreted by the ICM generic dialogue manager.

There are the following major abstraction mechanisms: input lexica, output lexica, input grammars, output grammars, semantic rules, APSG formalism for input lexica and input grammars, and state transition diagram formalism for the dialogue structure. Input and output grammars, semantic rules and the dialogue structure are described as GL diagrams.

The existence of two input lexicon/grammar formalisms are due to historical project organisations.

3.1 Input lexica

The input lexica are divided into a number of sub-lexica that are referenced in sub-grammars and a global input lexicon that are referenced in sub-grammars or outside grammars.

Each input lexicon L consists of an ordered list of lexicon entries E_{Li} which has a categorical reference point P_{Li} (denoted a pragmatic category) and consists of the tuple $E_{Li} = [P_{Li}, ev_{Li}, D_{Li}, I_{Li}, H_{Li}, V_{Li}]$ where ev_{Li} is a functional predicate reference, D_{Li} is a device reference and I_{Li} is an item such as a text string or a variable, H_{Li} is an action program, and V_{Li} is a declaration of semantic slot-fillings as a list tuples $[cat_{Lij}, val_{Lij}]$ where cat_{Lij} specify a slot in any SO (semantic object, see below), and val_{Lij} specify the value to be assigned to that slot.

Lexicon access is through the P_{Li} and involves evaluation of the predicate $ev_{Li}(D_{Li}, I_{Li})$ on the basis of an object representation describing the incoming event and possibly other context. A P_{Li} can occur in more than one E_{Li}. The H_{Li} program is executed only if the lexicon entry E_{Li} is used positively in the state transition algorithm (see below) and provide a mechanism for maintaining contexts.

3.2 Semantic objects

A semantic object SO represents a simple semantic value of input in terms of nested frame structures. The SO is defined as a list of slots $[name, type, default - value]$ where $default - value$ applies if the slot is not otherwise assigned.

3.3 Input symbols

Three different input symbols exist which reflects three abstraction levels of input. An input symbol is connected as a branch from a dialogue state, or it occur in input grammars.

3.3.1 Simple input symbols

A simple input symbol IS refers to a lexicon entry through the reference point P_{Li}, and that can be an abstraction of simple events such as a click on a button, a DTMF touch tone, or if an isolated word recogniser is attached, a single word. Further, it can be system generated events such as a time-out or an error.

3.3.2 Grammatical input symbols

A grammatical input symbol GIS refers to a context-free grammar. A grammar G is defined using the GL, a consists of terminals which are simple input symbols IS's, and non-terminals which are grammatical input symbols GIS's. Recursive grammar definitions are thus possible.

An input grammar specify the type of SO that represents the semantic value of the parsing results. Semantic values are assigned to slots of the SO by matching the cat_{Lij} of the lexicon entries referenced by the IS's of the grammar with slot names in the SO.

During runtime, the grammars are used by a recursive transition network (RTN) parser built into the dialogue manager.

3.3.3 Semantic input symbols

A semantic input symbol *SIS* refers to a semantic rule *R* that evaluates a semantic object *SO*. The branch of a semantic input symbol is taken if the rule it refers to evaluates to the largest scalar value in comparison with other rules.

A semantic rule *R* is represented as a GL diagram that evaluates an *SO* parameter and returns a scalar value (or nothing if the rule fails). The rule may access contextual data structures for the evaluation. A semantic rule declaration specify an *SO* type and a list of APSG sub-grammars that map semantic values on that *SO*.

As mentioned above, the dialogue manager and the APSG parser are separate modules that interact through a simple interface. The dialogue manager gives the APSG parser the following information: a list of *SO*'s, a list of names of APSG sub-grammars, and the textual representation of an utterance as determined by a speech recogniser. The parser parses then utterance according to the set of sub-grammars and maps semantic values on the *SO*'s.

3.4 Dialogue states

The DDL formalism experimentally allow multi threaded dialogue where the dialogue may be in several states concurrently. However, for the sake of clarity in this paper, there is only one current state. In this state, branches to input symbols describe the input events that are expected to occur while in that state. Input symbols specify whether input events are treated as ordinary input or exceptions.

3.5 Handlers

Handlers are branches to input symbols from either the *process start* symbol (which is the start of the dialogue description) or from *procedure start* symbols (which are the start of sub-dialogues). A handler *H* is *attentive* when the *process start* symbols or *procedure start* symbol from which it is a branch has been executed and the corresponding *process stop* or *procedure stop* symbols has not been executed (i.e. it is *attentive* while the sub-dialogue in which it is declared is active). An *attentive* handler can be invoked.

Handlers are either *reverting* or *non-reverting*. A *reverting* handler eventually transfers control to the dialogue state in which it was first invoked. A *non-reverting* handler transfers control to any position in the dialogue description in which case the dialogue manager restores its context to the context for that position.

3.6 State transition algorithm

The state transition algorithm specifies what actions are taken by the dialogue manager while waiting in a dialogue state and an input event is arriving. Before the dialogue manager enters the waiting state it computes the constraints and imposes these on the input devices. In addition, the relevant parts of the constraints together with the set of *SO*'s are handed to the APSG parser (which then prepares itself for parsing). When input arrives, the dialogue manager performs the following actions

1. If RTN parsing status is set, continue parsing with the input event as the next input token. If parsing completes, select the branch corresponding the the input grammar that was used to complete the parse.

2. If input is classified as a continuous speech utterance invoke the APSG parser and retrieve the *SO*'s.

3. If any *SO*'s have filled slots, then for each of these *SO*'s evaluate the semantic rule that knows about that *SO*.

4. If any rule evaluates to a value, then select the branch corresponding the rule that evaluates to numerically highest scalar value and terminate the state transition algorithm.

5. Invoke the RTN parser with the input event as the first token. If parsing is possible according to any input grammar, then set the RTN parsing status and terminate the state transition algorithm.

6. Evaluate each of the *IS*, and select the branch of the first one that evaluates to true and terminate the state transition algorithm. The order of evaluation is such that for all E_{Lx} referenced by the *IS*'s, E_{Li} is evaluated before E_{Lj} if $i < j$.

7. For each attentive handler, considered in reverse order of becoming attentive, evaluate the *SIS*, *GIS*, and *IS* for that handler as specified in steps 3 to 6.

When the state transition algorithm terminates, one of the following actions are taken

- If any branch was selected, transfer control to that branch.

- If RTN parsing status is set, calculate new constraints and impose them on the input devices.

4 Calculation of constraints

The constraints imposed on the input devices are calculated as follows: Let S_i denote a dialogue state i, H_j a handler j, W_{S_i} the set of simple input events expected in dialogue state S_i, W_{H_j} the set of simple input events expected by handler H_j, G_{S_i} the set of input grammars expected in dialogue state S_i, G_{H_j} the set of input grammars expected by handler H_j, SO_{S_i} the set of semantic representations expected in dialogue state S_i, and SO_{H_j} the set of semantic representations expected by handler H_j.

We use the term *expected* to cover both ordinary input and exception input. Further, let $\mathcal{F}(G, w_1 w_2 \ldots w_n)$ denote the set of possible next terminal symbols for the grammar set G given that the n terminals $w_1, w_2, \ldots w_n$ has been successfully parsed, and $\mathcal{A}(SO)$ be the set of APSG sub-grammars that map semantic representation on SO. The constraints in dialogue state S_i given that the handlers $H_1, H_2, \ldots H_m$ are attentive are expressed as C_W which are the simple events constraints (single words, buttons etc.) and C_A which are the APSG sub-grammars that are active.

The constraints are calculated as follows: if RTN parsing status is not set (i.e. $n = 0$), then

$$\begin{aligned} C_W &= W_{S_i} \\ &\quad \cup \cup_{j=1}^{m} W_{H_j} \\ &\quad \cup \mathcal{F}(G_{S_i}, \varepsilon) \\ &\quad \cup \cup_{j=1}^{m} \mathcal{F}(G_{H_j}, \varepsilon) \\ C_A &= \mathcal{A}(SO_{S_i}) \cup_{j=1}^{m} \mathcal{A}(SO_{H_j}) \end{aligned}$$

If RTN parsing status is set (i.e. $n \neq 0$), then

$$\begin{aligned} C_W &= \mathcal{F}(G_{S_i}, w_1 w_2 \ldots w_n) \\ &\quad \cup \cup_{j=1}^{m} \mathcal{F}(G_{H_j}, w_1 w_2 \ldots w_n) \end{aligned}$$

C_A is not calculated when RTN parsing status is set.

To yield the I_{Li} the W_{S_i} is mapped through the union of all E_{Li} that have the reference point P_{Li} that are referenced in all IS's branching from the dialogue state S_i. The I_{Li} for W_{H_j} is derived similarly to W_{S_i}.

The sets G_{S_i} and G_{H_j} are dynamically and temporarily reduced during parsing as grammar rules may fail.

The dialogue manager statically computes all W_{S_i}, W_{H_j}, $\mathcal{A}(SO_{S_i})$ and $\mathcal{A}(SO_{H_j})$, except for those sets that depend on variables. These are computed dynamically. The constraints are imposed on the input devices as Δ sets (which reduces the communication overhead between modules). A device that adhere to the GDS protocol has the notion of constraints sets C_i that are enumerated by the dialogue manager and a special active unordered constraints set C_a that constitute the actual constraints on input, and a stack of constraints sets. The dialogue manager maintains the sets for all devices independently. A device accepts the constraints related commands:

$C_i = I_{L1} I_{L2} \ldots I_{Lk}$
$C_a = C_a \cup C_i$
$C_a = C_a \cup \{I_{L1} I_{L2} \ldots I_{Lk}\}$
$C_a = C_a \backslash C_i$
$C_a = C_a \backslash \{I_{L1} I_{L2} \ldots I_{Lk}\}$
$C_a = \emptyset$
$push(C_a)$
$pop(C_a)$

5 Conclusion

The dialogue system platform described in this paper has been applied for a number of prototype dialogue systems and the input constraining method has been used for isolated word recognisers and continuous speech recognisers.

The calculation of constraints is automatically performed by the dialogue manager, and is therefore not part of a high level dialogue description.

In the flight ticket reservation dialogue system the constraining of the speech recogniser which have limitations on vocabulary size and language model size has allowed dialogue structures that would otherwise require limitations of the speech recognisers of about 5 times larger. The use of constraints thus allows higher complexity dialogues.

References

[Baekgaard 1995] Anders Baekgaard: *A Platform for Spoken Dialogue Systems*, Proceedings from ESCA Tutorial and Workshop on Spoken Dialogue Systems, Vigsoe, June 1995, pp. 105-108.

[Dalsgaard and Baekgaard 1994] Paul Dalsgaard and Anders Baekgaard: *Spoken Language Dialogue Systems*, Proceedings in Artificial Intelligence, Volume "Prospects and Perspective in Speech Technology", edited by Chr. Freksa, Infix, München, September 1994, pp. 178-191. 1994.

[Music 1993] Bradley Music and Claus Povlsen: *The NLP Module of a Spoken Language Dialogue System for Danish Flight Reservations*, Proc. EuroSpeech conference 1993, Berlin, pp. 1859-1862.

[Young 1990] Sharyl R. Young: *Use of Dialogue, Pragmatics and Semantics to Enhance Speech Recognition*, Speech Communication 9, 1990, pp. 551-559.