Constraint Manipulation and Visualization in a Multimodal Dialogue System

Joakim Gustafson[1,2], Linda Bell[1,2], Johan Boye[1], Jens Edlund[2] & Mats Wirén[1]

[1]*Telia Research, Vitsandsgatan 9, S-12386 Farsta, Sweden*

[2]*Centre for Speech Technology, Drottning Kristinas väg 31, S-10044 Stockholm, Sweden*

Abstract

When interacting with spoken and multimodal dialogue systems, it is often difficult for users to understand and influence how their input is processed by the system. In this paper, we describe how these problems were addressed in the multimodal real-estate dialogue system AdApt. During the course of a dialogue, the user's contraints are translated into symbolic icons that are visualized on the screen and can be manipulated by drag-and-drop operations. Users are thus given a clear picture of how their utterances are understood, and are given a transparent means of controlling the interaction with the system.

1. INTRODUCTION

Spoken dialogue and multimodal dialogue interfaces are increasingly being used as intelligent front-ends to databases. Such interfaces allow non-experts to solve complicated search problems, as the user has access to spoken natural language and graphical operations instead of having to learn a query language such as SQL. Furthermore, the user can incrementally refine and modify his search queries by means of an ongoing dialogue with the system, rather than having to construct entire queries in one go. This article describes how such search problems are handled in AdApt, a multimodal dialogue system. AdApt lets the user look for apartments for sale in downtown Stockholm, Sweden. The apartment domain is complex, with a large number of factors that can be independently varied (from the user's point of view), such as price, monthly fee, size, number of rooms, location, the presence/absence of things such as an elevator, a balcony, fireplace, or tiled stove in the apartment, and so on.

The structure of the resulting dialogue is both surprisingly simple and potentially very complex, depending on the level one looks at. At the lowest level, the vast majority of single utterances correspond to straightforward database queries in the sense that users ask specific questions regarding concrete properties, one at a time. In contrast, people rarely ask general questions involving subjective criteria that have non-trivial mappings to database queries, such as “I want a light and cozy apartment in a quiet surrounding”. Thus, the level of single utterances typically does not exhibit great complexity. In practice, this means that processing can be done with a shallow parser and, furthermore, that the relation between semantic representation and database query is straightforward and has a direct mapping.

At the level of continuous dialogue, however, it is clear that the succession of simple queries just provides a means to fulfill a much more complex task that rather corresponds to browsing: Users rarely come with an a priori set conception of what they are looking for; rather, they refine and modify their stated needs in an incremental fashion in the course of the dialogue, partly depending on what apartments are available, and partly depending on new features that spring to mind based on previous examination of objects.

Thus, to allow users to properly navigate the database, the possibility of both adding and retracting constraints is essential. In particular, if the set of solutions to a specific query is large, the user may want to add constraints to limit the solution set, instead of examining the members of the larger set one by one. If the set of solutions is empty, on the other hand, the system might automatically retract constraints until a solution is found (unless this task is left to the user). However, as the user and system keep retracting and adding constraints, the current set of constraints becomes less and less transparent. To efficiently convey the current set of constraints to the user, thereby achieving the necessary grounding, AdApt takes advantage of multimodality by visualizing the current search constraints as small icons. The user can retract constraints either by dragging the corresponding icon and dropping it in the “Trashcan”, or by pointing to an icon and saying ”forget this” or some similar phrase, for example: “If you forget about the balcony, what apartments can you find?”.

In this paper, we will describe how the AdApt system was incrementally developed to handle such advanced constraint manipulation and visualization.

2. BACKGROUND

Previous studies have emphasized the problem of extracting useful information from large multidimensional search spaces (Burke et al.,1997; Pu et al., 2000). However, in contrast to our system, the ‘assisted browsing’ strategy of Burke et al. (1997) aims at trying to make the user avoid specific (concrete) questions by instead presenting suggestions and examples, thereby leading the user’s search in a direction determined by the system. Pu et al. (2000) describe a travel-planning application in which user criteria and preferences are modelled as constraint satisfaction, using a range of overview displays. Although this seems primarily useful in situations where users have well-structured a priori ideas of what they are looking for, it is an interesting approach that might well be adopted as a complementary strategy.

For users of spoken dialogue systems, understanding how or even if your input to the system is being processed can be difficult. Telephone-based information retrieval systems often require users to provide repeated confirmation prompts as a way of giving feedback on the previous turn and ascertaining that the dialogue is on the right track. When a problem occurs in the dialogue, the system has to reprompt the user by requiring her to repeat or rephrase the previous turn. However, users find sequences of explicit confirmation prompts and reprompting awkward and tedious. Studies have shown that repeated confirmation turns give users the impression that the system is slow and make the human-computer dialogue appear less natural (Boyce, 1999).

Multimodal dialogue systems, with which the user and system can interact with two or more modalities, offer to provide the solution to some of the problems facing speech-only systems. As reported by Lamel et al (2000), spoken dialogue systems with multimedia interfaces can efficiently display all the user’s options on the screen instead of conveying this information by means of dialogue. Similarly, a multimodal system can provide its users with feedback without requiring a separate dialogue turn by giving graphical indications of its internal state. Previous studies have shown that an interface which combined pen and voice was perceived as more efficient than one which used either speech (Oviatt, 1997) or graphics (Cohen et al, 1998). Furthermore, users of multimodal systems tend to switch from one input modality to another if the system fails to understand an initial request (Oviatt and VanGent, 1996).

According to the theory of grounding (Brennan and Clark, 1996; Clark and Wilkes-Gibbs, 1986), human discourse should be viewed as a joint activity where speakers try to establish a common ground between them. Speakers must continuously make sure that their utterances are received they way they intended by the other participants in the conversation. However, which criteria that are actually required for grounding vary with the context and situation (Clark and Wilkes-Gibbs, 1986; Clark and Schaefer, 1989).

In human-computer interaction, grounding becomes even more important while the process itself becomes more complicated. Complex dialogue systems which allow users to manipulate many different constraints in the course of their interaction are especially demanding from this point of view. Brennan (1998) argues that errors that occur in human-computer interaction often are caused by failures in grounding, since the system and user lack enough evidence to coordinate their different knowledge states.

In a recent study, it is shown that presenting users with a graphical representation of the discourse domain and dialogue state of a spoken dialogue system can be useful (Terken and te Riele, 2001). The study reports on an experiment in which a unimodal version of a dialogue system was compared to a multimodal version of the same system. The multimodal version of the system was rated as being advantageous from the point of view of efficiency and user satisfaction. In the present study, we discuss how the feedback strategies of the multimodal AdApt system were improved, so that users of the system could get a better understanding of how their input was being processed. The idea was to increase the transparency of the system's internal representations and decisions by visualizing user constraints.

3. THE DEVELOPMENT OF THE ADAPT SYSTEM

AdApt is a Swedish dialogue system that was developed at the Centre for Speech Technology (CTT), with Telia Research as industrial partner (Gustafson et al. 2000). AdApt is a multimodal research system that allows the user to look for an apartment for sale in downtown Stockholm, Sweden. The apartment domain was chosen because it interests a lot of people, is complex enough, and encourages multimodal interaction. The system features a 3D-animated head, developed at KTH, which produces lip-synchronized synthetic speech (Beskow, 1997). Information about the location of retrieved apartments is displayed on an interactive map. The system makes a combined interpretation of the graphical input and the textual output from the speech recognizer, and sends the result to the dialogue manager.

The system was iteratively constructed with both simulated experiments and user tests with a fully implemented system. During the development of the system, the need to give the users means to manipulate the current search constraints became apparent: This led to the generation of the icon handler that visualizes the user's constraints on the screen.

3.1 Wizard-of-Oz experiments

Initially, a Wizard-of-Oz version of the system was developed. The simulated system accessed a database of genuine apartments for sale, automatically extracted from a commercial web site. The users primarily interacted verbally with the animated talking agent, but they could also provide graphical input by drawing areas or by selecting apartment icons on an interactive map. A human operator simulated the system's key functionalities, i.e. the analysis of the user's verbal and graphical input, dialogue management and multimodal response generation. Even though the system's verbal and graphical output was generated by means of ready-made templates, the human acting as wizard used his own intuitions to handle the turn-taking. The animated agent displayed a 'listening' gesture while the user was talking and then turned to a 'thinking' gesture when silence was detected. 32 subjects were given tasks that involved finding apartments with certain criteria given to them via pictorial scenarios. The analysis of the users' interactions was used to build grammars for the recognizer and parser as well as in the design of the dialogue manager. Furthermore, the analysis of the database revealed a large number of fragmented utterances. Most of these utterances either consisted of feedback on the system's previous turn followed by a request or topicalized references to an apartment followed by a question about the same. To enable the system to make real-time decisions on how to handle these fragmented user utterances, we introduced an Input/Output manager to manage the information flow of the system. This I/O manager only sends the user utterances to the dialogue manager that the parser considers to be complete (Bell et al, 2001).

3.2 Pilot study

An early version of the fully implemented system was tested as part of a bullet course in speech technology held at CTT. 15 employees of the CTT industrial partners interacted with the system in groups of three. Apart from the 'thinking' gestures that had been used in the WoZ experiments, the animated agent also generated 'continued attention' gestures when the parser had judged an utterance as being non-complete. The analysis of these interactions showed that it was difficult for the users to interpret some of the turn-taking gestures and that they also expressed uncertainty as to which search constraints the system was using at any given time. These problems lead to a tendency for some users to resort to a command-like language. The data from the pilot study was used to upgrade the system's input modules, dialogue manager and turn-taking gestures. The study also made it obvious that it would become necessary to give the users continuous feedback on the system's search constraints.

3.3 User study

The upgraded AdApt system was used to investigate the efficiency of two different feedback strategies to communicate whether the system was ‘listening’ or ‘working’. The first version showed an hourglass to signal that the system was preparing a response. The second version used updated facial gestures for ‘continued attention’ and ‘thinking’. In the user study, a version without any visual feedback to support turn-taking was also used as a baseline. Each of the three versions was tested by eight users, adding up to a total of 24 subjects. The subjects did not receive any task other than “try to get information about apartments that you are interested in”, and were not informed about the capabilities of the system. However, they were told how to start a new session in case they got stuck. Each subject interacted with the system for more than 30 minutes, and the total number of subject utterances amounts to more than 3000.

An analysis of the effect of the system’s feedback strategies for turn-taking is reported in another paper in these proceedings (Edlund and Nordstrand 2002). It was interesting to note that several users explicitly asked the system to give them feedback on its current search constraints or told it to remove or change some of its constraints.

4. CONSTRAINT MANIPULATION

To allow smooth information browsing, the system has to modify, refine and relax the user's constraints, sometimes in non-obvious ways. In a typical dialogue, the user begins by stating some constraints on the apartment he wants to buy. The system prompts for more constraints, while at the same time showing the current set of apartments in the database matching the search constraints as colored icons on the map. The user then either gives more constraints (e.g. “I want to live in the Old Town”), which the system adds to the search query, or starts asking questions about specific apartments, e.g. “How much does the red apartment cost?”.

As long as the user keeps adding constraints and the resulting set of apartments is non-empty, the system’s actions are straightforward. A first complication arises when the user revises the query. For instance, he might think that the presented apartments are too expensive, and change or relax some constraints by saying “Look for two-room apartments instead” or “Forget about the balcony”. Obviously, the current search query changed as a result of this utterance, but the question is *how*. There are several possible strategies:

- ***Minimal change.*** Just modify the constraint that the user is explicitly referring to, and keep everything else. In the first example above, this would make the system change the *number_of_rooms* attribute from 3 to 2. One quickly realizes that this strategy is untenable, since various attributes can be related to each other (e.g. the *street* and *area* attributes are obviously related in the apartment domain).

- ***Maximal change.*** Throw everything away except for the modified constraint. In the first example above, this would make the system look for any two-room apartments. This strategy tends to be frustrating for the user, especially since a recognition error can make the system throw out all accumulated constraints.
- ***Minimal change with dependencies.*** Modify the constraint the user is explicitly referring to, and everything that is related to it. For instance, if the user modifies the value of the *area* attribute by saying ``Look in the Old Town instead", the system also clears the *street_name*, *x-coordinate* and *y-coordinate* attributes. This strategy leads to some tricky considerations; e.g. there is clearly a relationship between the size of an apartment (in square meters) and the number of rooms, but this relationship is not very obvious.

A related situation which is not too uncommon is that the set of database matches is empty. This situation can arise due to misrecognitions (e.g. “a one-room apartment” is understood as “a ten-room apartment”), or because the user has unrealistic expectations on the offering of apartments for sale. To get out of this dead end, the system has to relax or retract some constraints from the current query. Again there are several possible strategies:

- ***Retract the last constraint.*** The most obvious strategy is to remove the constraints added by the last user utterance (after all, adding those constraints resulted in an empty answer set). The drawback is that this strategy does not give the user much guidance on how to continue.
- ***Relax numerical constraints.*** A straightforward strategy is to relax exact numerical values to intervals, and to increase the size of intervals. To some extent this strategy is absolutely necessary, e.g. instead of searching for apartments that cost two million, the system must search in a suitable interval around two million.
- ***Importance ranking.*** The system tries to guess what attributes are the most important for the user, and throws out everything but the values of those attributes. For instance, if the user has stated some desired properties without having been prompted, the system might guess that these constraints are the most important for the user.

- ***Minimal balanced relaxation.*** The system relaxes as few constraints as possible in the search query, until the set of matches in the database is non-empty. The greatest drawback of this strategy is that it is complicated to feedback to the user how the search query has been modified (see more next section).
- ***Remove unreasonable constraints.*** The system uses domain-dependent rules to detect and remove unreasonable constraints (which usually are due to recognition errors), like apartments with twenty rooms.

In fact, it is advantageous to let advanced users control the system's constraint relaxation strategy. In this domain, a user might want to distinguish between necessary constraints (e.g. the apartment cannot cost more than two million) on the one hand, and constraints that express desirable rather than necessary properties on the other hand (e.g. "balcony" or "top floor"). The constraint visualization strategy presented in the next section allows an elegant means of making this distinction. Each constraint icon can be provided with a 'lock' button. By clicking on this lock button, the user can effectively instruct the system not to relax that particular constraint, regardless of the relaxation strategy employed.

5. CONSTRAINT VISUALIZATION

Advanced constraint manipulation is needed to facilitate browsing. However, to prevent the user from feeling a lack of control, the system must inform the user of what it is doing According to Schneiderman (1997), a user-friendly computer interface must be comprehensive, predictable and give the user the notion of being in control. To do this, the system should be reactive and continuously give the user feedback on what it has received. The user wants to know that her input was correctly understood and that the system will perform the intended task. As discussed above, however, speech-only systems that use repeated confirmation prompts are perceived as tedious and unnatural (Boyce, 1999).

In the development of the AdApt system, a multimodal confirmation strategy was used to signal what the system has received and understood. This approach was chosen to make the dialogue system appear reactive, and the dialogues more natural and less system-directed. Immediately after a user has finished an utterance, the system responds with feedback by showing icons that represent the information units in the recognized and parsed string. The system also marks the locations of the apartments on a small map. To indicate that the initiative remains with the user, and that she is free to either add more constraints or start asking about a specific apartment, the animated agent shows a facial gesture that encourages the user to continue speaking. If the user still does not speak within a timeout-period of about a second, the system takes the initiative by asking the user for one of the feasible constraints that she has not yet supplied.

The system also displays its *inner state*, i.e. all the constraints presently used to perform a query. If the system decides to relax some of the user's current constraints before conducting its search, the inner state contains something different than what the user actually said. If, for instance, the user requested an apartment for 2 million the system will actually look for apartments in the price range of 1.5 and 2.5 million. In this case, the system's decision to relax the price constraint could be wrong. The user might have intended for the system to search for apartments that cost *less* than 2 million. While this misunderstanding would have been quite tedious to ground verbally, it is easier to do graphically. The price constraint is showed graphically with an icon that indicates the price range. This can be done either with a scale, which is highlighted between 1.5 and 2.5 million, or with an icon like the one shown in Figure 1 below. The user can select the icon to change the price range graphically or verbally.

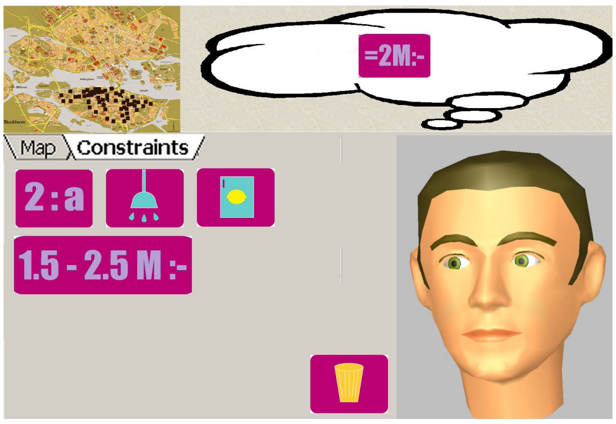

Figure 1. In the agent's thought balloon, the currently recognized constraint "two million" is visualized. In the bottom left corner, illustrating the system's inner state, there is instead an icon representing the relaxed constraint 1.5-2.5 million.

Visualization of constraints also facilitates browsing in the data set. If the user had to supply a large number of constraints to get a graspable number of apartments, it could be difficult to remember under what conditions the search result was retrieved. In AdApt, the constraints can be displayed graphically and are thus easily remembered. This approach also makes it possible for the user to change a specific constraint multimodally. For instance, the room-constraint can be removed either by dragging the icon to the trashcan or by clicking on it and saying “forget this” or changed by saying “two rooms”. Constraint visualization will also facilitate error recovery during human-computer miscommunication. During error recovery in spoken dialogue systems, user utterances tend to get long, complex and unpredictable. Such utterances are difficult for the system’s recognition grammar to handle. Instead of saying something like “No no no, didn’t you hear me, I said I wanted two rooms not ten rooms”, users of AdApt will have the possibility of selecting the room icon and saying “two”.

6. ENABLING VISUALIZATION OF CONSTRAINTS

Certain aspects of the AdApt system architecture aspects facilitated the visualization of the inner state of the dialogue manager. This allowed the system to give the user graphical or verbal feedback throughout the dialogues. The system is able to give feedback on the previous turn and then wait for a user reaction before initiating the next turn. In the following section, we describe the aspects of the system necessary to visualize the system’s inner state, as well as the modifications necessary to implement the Icon handler.

6.1 The system architecture

The AdApt system is designed to facilitate user studies in multimodal settings. The system architecture makes it relatively easy to add system modules that provide new information channels. For the same reason, the communication protocols between the modules need to be easily extendable. Furthermore, the system is developed at two different sites, which makes flexibility regarding platforms and programming languages important.

The system is modular and distributed, with each module running in a separate process. The modules communicate via a broker using TCP/IP-sockets, which works well when the system runs on a single computer, but also allows the system to run in a distributed manner over a number of computers. The modules can be executed either under Windows or Unix.

Another facilitating feature was the usage of XML encoded messages for the communication between modules. This made it easy to add new information parts in the messages without having to rewrite all other modules. XML also has some other advantages: It is fairly standardized, it is not tied to any particular platform or language, and it is readily transformable and expandable. The bulk of the AdApt system is coded in Java, Prolog and Tcl, all of which are languages that have standard libraries for parsing and building XML messages. The choice of XML for the inter-process communication was fairly straightforward. However, the question of which modules to include is still an open one. Apart from the standard dialogue modules, i.e. speech recognition, parsing, etc., two additional modules are used in the AdApt system: the I/O manager and the GUI manager.

6.2 Robust Parser

The system uses a two-phase robust shallow-processing parsing algorithm to produce the semantic representation of utterances. In its first phase, the parser scans the string of words from left-to-right, and the sequence of graphical events in time-order, collecting a set of indicators triggered by syntactic patterns. For instance, the word “Hagagatan” would produce indicators that the user is talking about a street, that this street is most likely part of an address; thus the user is implicitly referring to an object that has an address, and since apartments are (currently) the only known kind of objects that have addresses, the user is implicitly referring to an apartment.

In the second phase, the parser uses heuristics to weigh all this information together, determining the utterance type (wh-question, yn-question, acknowledgement...), what the sought object is (an apartment, a price,...), the appropriate values of database slots, and the referential information expressed in the utterance. The final output of the parser is a sequence of semantic expressions of the form previously described in Bell et al (2001). By parsing verbal fragments and graphical input together, the system effectively interprets multimodal user input like “I want to live here” accompanied by the drawing of a region on the map, or “Forget that” accompanied by a drag-and-drop operation placing a constraint icon in the trashcan.

6.3 Dialogue Manager

The dialogue manager performs three main tasks: (1) Contextual interpretation of the user's utterance (2) Dialogue act classification, and (3) Response generation.

Contextual interpretation involves a number of steps. The explicit references detected by the parser are resolved, elliptic answers or questions are interpreted and domain-dependent re-interpretation rules are applied. For instance, the elliptic answer “two” is understood as “I can pay two Swedish crowns” if the system's latest question was “How much do you want to pay for your apartment?” where after “two Swedish crowns” is re-interpreted as “two million Swedish crowns”. In general, the contextual interpretation process might generate a set containing several hypotheses representing possible interpretations of the user's utterance.

After contextual interpretation, the dialogue manager will produce one verbal and one symbolic feedback of the user's utterance, as well as a symbolic representation of the current search constraints used by the system. This information is then sent to the I/O manager.

The dialogue manager then heuristically classifies each of its hypotheses as being a certain dialogue act, whereafter the winning hypothesis is selected by a voting procedure. This classification effectively decides in what way the constraints C expressed by the user's latest utterance should be combined with the set A of previously accumulated constraints. For instance, if the user's utterance is found to be a "preference", C and A are conjoined; in the case of a "new_preference", A is replaced by C, and in the case of a "change_preference", parts of A are replaced by C, etc. The resulting set of constraints will then be modified using the strategies described in section 4.

Finally, the dialogue manager generates the system's response utterance; paraphrases of the search results, prompts to give more constraint restricting the solution set, clarification questions, and so on.

6.4 I/O Manager

The I/O manager was implemented in order to facilitate flexible turn-taking in the dialogue and to make it possible to merge different input modes. It also handles the timing in the system. It decides which bits of input go together (is a click a part of the following utterance or the previous one?) and how to coordinate verbal and graphical output. In addition it handles timeout in the system, for instance how long to wait after giving feedback on the previous user utterance before asking the next system question, thus giving the user time to react on the feedback.

The I/O manager is responsible for merging input from different channels into a multimodal message that it sends to the parser and dialogue manager. It also decomposes the multimodal output message from the Dialogue manager into commands that is sent to the different output modules.

The output message from the Dialogue manager was extended with two new parts: A feedback part and a constraint history part. In turn, the feedback part consists of three subparts: A verbal paraphrase of the constraints in the user's latest utterance, a list of names on icons representing these constraints, and a list of apartments found using the current constraints. The I/O manager assesses the recognition confidence score to decide which kind of feedback strategy to use. However, it will always tell the Map handler to display the current search results as dots on the small overview map. Hopefully, this system behaviour will allow for less restricted dialogues, where the user can decide in each turn to keep the initiative by adding more constraints or start asking about the found apartments, or he can give the initiative to the system by waiting for it to ask for more constraints.

6.5 GUI Manager

The GUI manager provides a common frame for the different output modules (animated talking head, map handler, icon handler). The system consists of a number of separate processes, something which is completely hidden from the user by this module. The GUI manager creates a main window with a certain layout, and then lends parts of the windows to the other modules. A function that makes it possible for all GUI modules to share the same keyboard input is included, as is the possibility to have different modules share the same GUI space by using tabbed windows. In the present implementation, the Map handler and the Icon handler share one tabbed window, but they both have separate additional windows that are always visible.

6.6 Icon Handler

The Icon handler visualizes the graphical feedback generated by the Dialogue manager, in order to ground the constraint manipulations of the system. In the current version of the system, the graphical feedback is somewhat crude. Although the system distinguishes between the utterances “I’d like an apartment with a balcony” and “Does that apartment have a balcony?” (which would result in completely different answers from the system), both utterances would be paraphrased with a “balcony” icon. Thus, the pragmatic function of the utterances are not paraphrased, only the propositional contents. In order to convey pragmatic functions as well, more complex icons will needed. At this point, however, it is not clear whether the use of such complex symbols would really improve the human-computer interaction in a dialogue system.

Icons are a central part of today’s graphical interfaces, their purpose being to remind the user of basic functions and to give the user a simple access to the same. The most important design criteria when developing icons is that they should be easy to recognize (Martin and Eastman, 1996). However, they do not necessarily have to resemble what they represent. It is common to use the ‘metaphor paradigm’ when designing icons. A problem with using metaphors is that they do not scale well to more complicated functions and it can also be hard to find a logical metaphor. In the ‘idiomatic paradigm’ the user instead learns to connect certain icons to functions in the same way that people use idioms in language (Cooper 1995). This paradigm is used in road signs where the driver unconsciously learns and uses road signs all the time without having to actively think about it.

There are two reasons for using icons to give feedback on what a dialogue system has understood in the user’s utterance. Firstly, since graphical icons are non-linguistic and do not use the verbal channel they do not increase the users' cognitive load. Secondly, users are good at automatically picking up the meaning of iconographic symbols if they are presented to them repeatedly while they are doing other things.

In the development of AdApt, the most important criterion was that the graphical symbols should be easy to interpret and separate from each other. Since the learning curve for abstract icons is longer, it seemed preferable to develop concrete icons that could be used in shorter user studies. Below are examples of icons with different degrees of abstraction that we have considered:

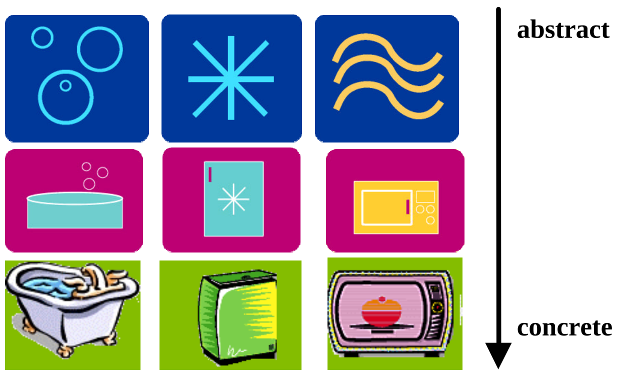

Figure 2. Examples of icons of bath tub, freezer and microwave oven with different degrees of abstraction.

Users can either manipulate the icon constraints graphically by dragging them to a trashcan or multimodally by selecting them at the same time as speaking. The Icon handler will in both cases send the semantic meaning of the graphical operation to the I/O manager, which will join it with the output from the recognizer and then send this to the multimodal parser.

7. CONCLUDING REMARKS

The presented approach to constraint visualization and manipulation gives users a better insight into how the system has understood their input, as well as supplying them with a transparent means of controlling the interaction with the system. The modular, highly asynchronous architecture of the Adapt system enabled a smooth implementation of the ideas. Ongoing work includes a comparative user study, assessing the effectiveness of the proposed constraint visualization technique, compared to that of a more traditional verbal feedback method.

8. REFERENCES

Bell, L, Boye, J, and Gustafson, J. 2001. Real-time Handling of Fragmented Utterances, *Proceedings of NAACL*.

Beskow, J. 1997. Animation of Talking Agents, In *Proceedings of AVSP'97*, ESCA Workshop on Audio-Visual Speech Processing, Rhodes,Greece, September 1997.

Boyce, S. 1999. Spoken natural language dialogue systems: User interface issues for the future. In Gardner-Bonneau, (ed) *Human Factors and Voice Interactive Systems*, p. 37-62.

Brennan, S. E. 1998. The grounding problem in conversations with and through computers. In Fussell, S. R. and Kreutz, R. J. (eds.) *Social and Cognitive Psychological Approaches to Interpersonal Communication*, 201-225.

Brennan, S. E. and Clark, H. H. 1986. Conceptual pacts and lexical choice in conversation. *Journal of Experimental Psychology: Learning, Memory and Cognition* **22**(6):1482-1493

Burke, R. D., Hammond, K. J, and Young, B. J. 1997. The FindMe Approach to Assisted Browsing. *IEEE Expert*, **12**(4): 32-40.

Clark, H. H. and Schaefer, E. W. 1989. Contributing to discourse. *Cognitive Science* **13**: 259-294

Clark, H.H. and Wilkes-Gibbs, D. 1986. Referring as a collaborative process. *Cognition* **22**: 1-39

Cohen, P. R., Johnston, M., McGee, D., Oviatt, S. L., Clow, J., Smith, I. 1998. The efficiency of multimodal interaction: A case study. *Proceedings of ICSLP '98*

Cooper, A. 1995. *About face: The Essentials of User Interface Design*. New York: Hungry Minds

Gustafson, J, Bell, L, Beskow, J, Boye, J, Carlson, R, Edlund, J, Granström, B, House, D and Wirén M. 2000. AdApt – a multimodal conversational dialogue system in an apartment domain. *Proceedings of ICSLP*.

Edlund, J. and Nordstrand, M. 2002. Turn-taking gestures and hour-glasses in a multi-modal dialogue system, In this volume

Lamel, L., Rosset, S. and Gauvin, J-L. 2000. Considerations in the design and evaluation of spoken dialogue systems. *Proceedings of ICSLP*

Martin, A. and Eastman, D. 1996. *The User Interface Design Book*. New York: Wiley & Sons

Oviatt, S. 1997. Multimodal interactive maps: Designing for Human Performance. *Human Computer Interaction* **12**: 93-129

Oviatt, S. and VanGent, R. 1996. Error resolution during multimodal human-computer interaction. *Proceedings of ICSLP'96*, 204-207.

Pu, P. and Faltings, B. 2000. Enriching buyers' experiences: the SmartClient approach, in *Proceedings of ACM CHI'2000*, ACM Press.

Terken, J. and te Riele, S. 2001. Supporting the construction of a user model in speech-only interfaces by adding multi-modality. *Proc. of Eurospeech*, 2177-2180.

Schneiderman, B. 1997**.** Direct Manipulation for Comprehensible, Predictable, and Controllable User Interfaces. *Proceedings of IUI97*, 33-39.