MATCH: An Architecture for Multimodal Dialogue Systems

Michael Johnston, Srinivas Bangalore, Gunaranjan Vasireddy, Amanda Stent, Patrick Ehlen,
Marilyn Walker, Steve Whittaker, Preetam Maloor
AT&T Labs - Research
180 Park Ave, Florham Park
NJ, USA, 07932
johnston@research.att.com

1/31/02

Paper ID: P0346

Keywords: multimodal, understanding, generation, finite-state, mobile

Contact Author: Michael Johnston

Under consideration for other conferences (specify)? none

Abstract

Interfaces for mobile information access need to allow users to dynamically adapt their choice of modes according to their preferences, task at hand, and physical and social environment. This paper describes a multimodal application architecture that facilitates rapid prototyping of multimodal interfaces with flexible input and adaptive output. Our testbed application MATCH (Multimodal Access To City Help) provides a mobile multimodal speech-pen interface to restaurant and subway information for New York City. Finite-state methods for multimodal language understanding are employed to enable users to interact using pen, speech, or dynamic combinations of the two. A speech-act based multimodal dialogue manager provides support for mixed-initiative multimodal dialogue. Multimodal generation and text planning components enable the system to respond using synchronized multimodal presentations that combine dynamic graphics with text-to-speech and are tailored to the user's individual preferences. The development of this architecture is driven by ongoing scenario-based evaluation and data collection with users.

MATCH: An Architecture for Multimodal Dialogue Systems

Paper ID: P0346

Abstract

Interfaces for mobile information access need to allow users to dynamically adapt their choice of modes according to their preferences, task at hand, and physical and social environment. This paper describes a multimodal application architecture that facilitates rapid prototyping of multimodal interfaces with flexible input and adaptive output. Our testbed application MATCH (Multimodal Access To City Help) provides a mobile multimodal speech-pen interface to restaurant and subway information for New York City. Finite-state methods for multimodal language understanding are employed to enable users to interact using pen, speech, or dynamic combinations of the two. A speech-act based multimodal dialogue manager provides support for mixed-initiative multimodal dialogue. Multimodal generation and text planning components enable the system to respond using synchronized multimodal presentations that combine dynamic graphics with text-to-speech and are tailored to the user's individual preferences. The development of this architecture is driven by ongoing scenario-based evaluation and data collection with users.

1 Multimodal Mobile Information Access

In urban environments tourists and residents alike need access to a complex and constantly changing body of information regarding restaurants, cinema and theatre schedules, transportation topology and timetables. This information is most valuable if it can be delivered effectively while mobile, since places close and plans change. Mobile information access devices (PDAs, tablet PCs, next-generation phones) offer limited screen real estate and no keyboard or mouse, making complex graphical interfaces cumbersome. Multimodal interfaces can address this problem by enabling speech and pen input and output combining synthetic speech and graphics (See (André, 2002) for a detailed overview of previous work on multimodal input and output). Since mobile devices are used in situations involving different physical and social environments, tasks, and users, they need to be both flexible in input and adaptive in output. Users need to be able to provide input in whichever mode or combination of modes are most appropriate, and system output needs to be dynamically tailored so that it is maximally effective given the situation and the user's preferences. We present here our testbed multimodal application MATCH (Multimodal Access To City Help) and the general purpose multimodal architecture underlying it, that combines finite-state multimodal input processing, with a multimodal dialogue manager, multimodal generation, and text planning capabilities in order to enable flexible input and adaptive multimodal output. MATCH is a working city guide

Figure 1: MATCH running on Fujitsu PDA

and navigation system that currently enables mobile users to access restaurant and subway information for New York City (NYC). MATCH runs standalone on a Fujitsu pen computer (Figure 1), yet can also run in client-server mode across a wireless network.

The user interacts with a graphical interface displaying restaurant listings and a dynamic map showing locations and street information. They are free to give commands or reply to requests using speech, by drawing on the display with a stylus, or using synchronous multimodal combinations of the two modes. For example, they can request to see restaurants using the spoken command *show cheap italian*

restaurants in chelsea. The system will then zoom to the appropriate map location and show the locations of restaurants on the map. Alternatively, they could give the same command multimodally by circling an area on the map and saying *show cheap italian restaurants in this neighborhood*. If the immediate environment is too noisy or public, the same command can be given completely in pen as in Figure 2, by circling an area and writing *cheap* and *italian*.

Figure 2: Unimodal pen command

The user can ask for the review, cuisine, phone number, address, or other information for a restaurant or set of restaurants. The system responds with graphical callouts on the display, synchronized with synthetic speech output. For example, if the user says *phone numbers for these three restaurants* and circles a total of three restaurants as in Figure 3, the system will draw a callout with the restaurant name and number and say, for example *Le Zie can be reached at 212-206-8686*, for each restaurant in turn (Figure 4). These information seeking commands can also be issued solely with pen. For example, the user could alternatively have circled the restaurants and written *phone*. The user can also pan and zoom around the map. For example, they can say *show upper west side* or circle an area and say *zoom in here*.

Figure 3: Two area gestures

MATCH can also provide a summary, comparison, or recommendation for an arbitrary set of restaurants. The output is tailored to the user's preferences

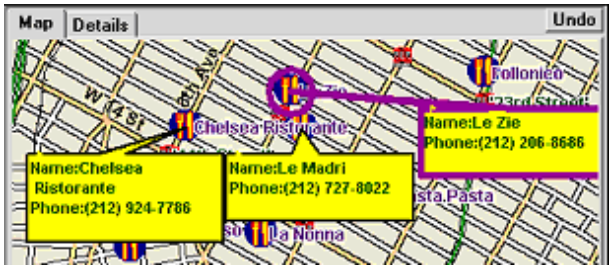

Figure 4: Phone query callouts

based on a user model, which is based on answers to a brief questionaire. For example, the user could say *compare these restaurants* and circle a large set of restaurants (Figure 5). The restaurants are ranked according to the user model and the top three are selected. If the user is oriented towards price first, and food quality second, the system would select restaurants by weighing price more highly than food quality in the ranking, resulting in an output such as:

Compare-A: *Among the selected restaurants, the following offer exceptional overall value. Uguale's price is 33 dollars. It has excellent food quality and good decor. Da Andrea's price is 28 dollars. It has very good food quality and good decor. John's Pizzeria's price is 20 dollars. It has very good food quality and mediocre decor.*

Figure 5: Comparing a large set of restaurants

The system also provides subway directions. For example, if the user says *How do I get to this place?* and circles one of the restaurants displayed on the map the system will ask *Where do you want to go from?* The user can then respond with speech e.g.: *25th Street and 3rd Avenue*, with pen by writing e.g. *25th St & 3rd Ave*, or multimodally: e.g. *from here* (with a circle gesture indicating location). The system then calculates the optimal subway route and dynamically generates a multimodal presentation indicating the series of actions the user needs to take. The system starts by zooming in on the first station

and then gradually zooms out graphically presenting each stage of the route along with a series of synchronized text-to-speech (TTS) prompts. Figure 6 shows the final display of a subway route heading south on the 6 train and transfering to the L train eastbound.

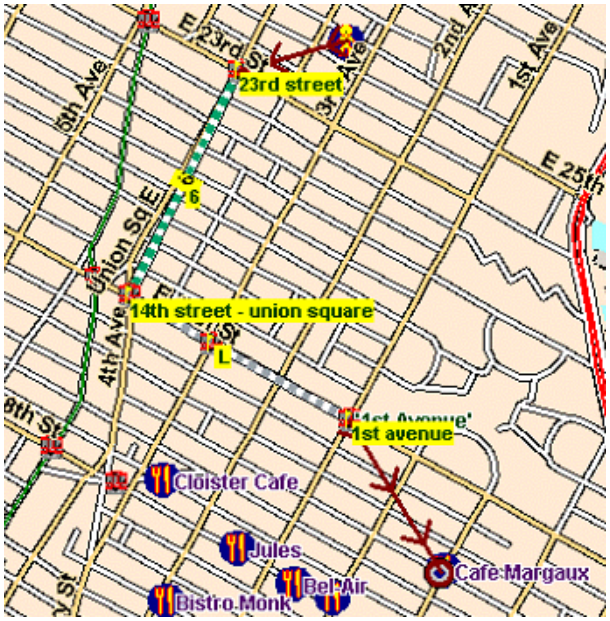

Figure 6: Multimodal subway route

The next section describes the architecture underlying MATCH and Section 3 describes the iterative evaluation and development process it enabled.

2 Multimodal Application Architecture

The multimodal architecture which supports MATCH consists of a series of agents which communicate through a facilitator MCUBE (Figure 7).

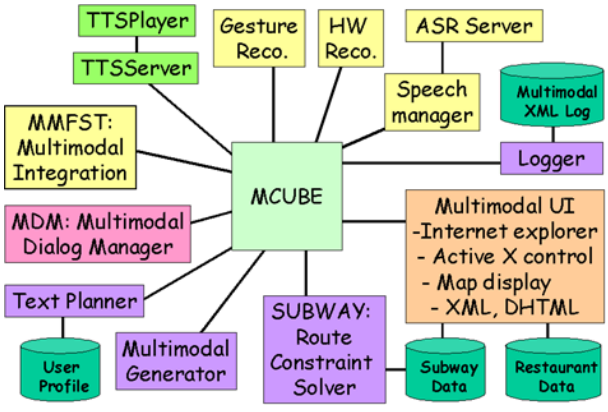

Figure 7: Multimodal Architecture

MCUBE is a Java-based facilitator which enables agents to pass messages either to single agents or groups of agents. It serves a similar function to systems such as OAA (Martin et al., 1999) and the use of KQML for messaging in (Allen et al., 2000). Agents may reside either on the client device or elsewhere on the network and can be implemented in multiple different languages. MCUBE messages are encoded in XML, which provides a general mechanism for message parsing and facilitates logging of multimodal exchanges.

Multimodal User Interface (Multimodal UI) Users interact with the system through the Multimodal UI, which is browser-based and runs in Internet Explorer. This greatly facilitates rapid prototyping, authoring, and reuse of the system for different applications since anything that can appear on a webpage (dynamic HTML, ActiveX controls etc.) can be used in the visual component of a multimodal user interface. A TCP/IP control enables communication with MCUBE.

For MATCH we utilize a control that provides a dynamic pan-able, zoomable map display. This control has been augmented with ink handling capability. This enables use of both pen-based interaction (on the map) and normal GUI interaction (on the rest of the page) without requiring the user to overtly switch 'modes'. When the user draws on the map their ink is captured and any objects potentially selected, such as currently displayed restaurants or subway stations, are determined. The electronic ink is broken into a lattice of strokes and passed to the gesture recognition and handwriting recognition components for analysis. When the results are returned the system combines them and the selection information into a lattice representing all of the possible interpretations of the user's ink.

In order to provide spoken input the user must hit a click-to-speak button on the Multimodal UI. This activates the speech manager described below. We found that in an application such as MATCH which provides extensive unimodal pen-based interaction, it was preferable to use click-to-speak rather than pen-to-speak or open-mike. With pen-to-speak, spurious speech results received in noisy environments can disrupt unimodal pen commands.

In addition to providing input capabilities, the multimodal UI also provides the graphical output capabilities of the system and coordinates these with text-to-speech output. When a request to display restaurants is received the XML listing of restaurants is essentially rendered using two stylesheets, yielding a dynamic HTML listing on the left and

a map display of restaurant locations on the right. When a request for the phone numbers of a set of restaurants is received from the multimodal generator the UI accesses the information from the restaurant database, then sends prompts to the TTS agent and, using progress notifications received through MCUBE from the TTS agent, displays synchronized graphical callouts highlighting the restaurants in question and presenting their names and numbers (Figure 4). These are placed using an intelligent label placement algorithm.

Speech Recognition A speech manager running on the device gathers audio and communicates with a recognition server running either on the device or in the network. The recognition server provides lattice output which is encoded in XML and passed to the multimodal integrator (MMFST).

Gesture and handwriting recognition Gesture and handwriting recognition agents are called on by the Multimodal UI to provide possible interpretations of electronic ink. Recognitions are performed both on individual strokes and combinations of strokes in the input ink lattice. For the MATCH application, the handwriting recognizer supports a vocabulary of 285 words, including attributes of restaurants (e.g. '*chinese*','*cheap*') and zones and points of interest (e.g. '*soho*','*empire*','*state*','*building*'). The gesture recognizer recognizes a set of 10 basic gestures, including lines, arrows, areas, points, and question marks. It uses a variant of Rubine's classic template-based gesture recognition algorithm (Rubine, 1991) trained on a corpus of sample gestures. In addition to classifying gestures the gesture recognition agent also extracts features such as the base and head of arrows. Combinations of this basic set of gestures and handwritten words provides a rich visual vocabulary for multimodal and pen-based commands.

Gesture and handwriting recognition enrich the ink lattice with possible classifications of strokes and stroke combinations, and pass it back to the multimodal UI where it is combined with selection information to yield a lattice of possible interpretations of the electronic ink. This is then passed on to MMFST.

The interpretations of electronic ink are encoded as symbol complexes of the following form *G FORM MEANING (NUMBER TYPE) SEM*. *FORM* indicates the physical form of the gesture and has values such as area, point, line, arrow. *MEANING* indicates the meaning of that form; for example an *area* can be either a *loc(ation)* or a *sel(ection)*. *NUMBER* and *TYPE* indicate the number of entities in a selection (*1,2,3, many*) and their type (*rest(aurant), theatre*). *SEM* is a place holder for the specific content of the gesture, such as the points that make up an area or the identifiers of objects in a selection (ids).

When multiple selection gestures are present an aggregation technique (Johnston and Bangalore, 2001) is employed in order to overcome the problems with deictic plurals and numerals described in Johnston (2000). Aggregation augments the gesture lattice with aggregate gestures that result from combining adjacent selection gestures. This allows a deictic expression like *these three restaurants* to combine with two area gestures, one which selects one restaurant and the other two, as long as their sum is three. For example, if the user makes two area gestures, one around a single restaurant and the other around two restaurants (Figure 3), the resulting gesture lattice will be as in Figure 8. The first gesture (node numbers 0-7) is either a reference to a location (loc.) (0-3,7) or a reference to a restaurant (sel.) (0-2,4-7). The second (nodes 7-13,16) is either a reference to a location (7-10,16) or to a set of two restaurants (7-9,11-13,16). The aggregation process applies to the two adjacent selections and adds a selection of three restaurants (0-2,4,14-16). If the user says *show chinese restaurants in this neighborhood and this neighborhood*, the path containing the two locations (0-3,7-10,16) will be taken when this lattice is combined with speech in MMFST. If the user says *tell me about this place and these places*, then the path with the adjacent selections is taken (0-2,4-9,11-13,16). If the speech is *tell me about these* or *phone numbers for these three restaurants* then the aggregate path (0-2,4,14-16) will be chosen.

Multimodal Integrator (MMFST) MMFST receives the speech lattice (from the Speech Manager) and the gesture lattice (from the UI) and builds a meaning lattice which captures the potential joint interpretations of the speech and gesture inputs. MMFST uses a system of intelligent timeouts to work out how long to wait when speech or gesture is received. These timeouts are kept very short by making them conditional on activity in the other input mode. MMFST is notified when the user has hit the click-to-speak button, when a speech result arrives, and whether or not the user is inking on the display. When a speech lattice arrives, if inking is in progress MMFST waits for the gesture lattice, otherwise it applies a short timeout and treats the speech as unimodal. When a gesture lattice arrives, if the user has hit click-to-speak MMFST waits for the speech result to arrive, otherwise it applies a short timeout and treats the gesture as unimodal.

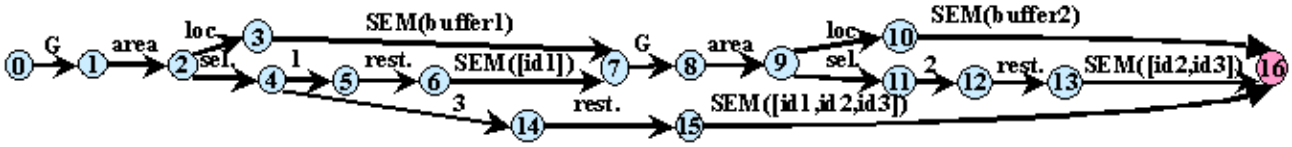

Figure 8: Gesture Lattice

MMFST uses the finite-state approach to multimodal integration and understanding proposed by Johnston and Bangalore (2000). In their approach, possibilities for multimodal integration and understanding are captured in a three tape finite state device in which the first tape represents the speech stream (words), the second the gesture stream (gesture symbols) and the third their combined meaning (meaning symbols). In essence, this device takes the speech and gesture lattices as inputs, consumes them using the first two tapes, and writes out a meaning lattice using the third tape. The three tape FSA is simulated using two transducers: *G:W* which is used to align speech and gesture and *G_W:M* which takes a composite alphabet of speech and gesture symbols as input and outputs meaning. The gesture lattice *G* and speech lattice *W* are composed with *G:W* and the result is factored into an FSA *G_W* which is composed with *G_W:M* to derive the meaning lattice *M*.

In order to capture multimodal integration using finite-state methods, it is necessary to abstract over specific aspects of gestural content (Johnston and Bangalore, 2000). For example, all the different possible sequences of coordinates that could occur in an area gesture cannot be encoded in the FSA. We employ the approach proposed in (Johnston and Bangalore, 2001) in which the gestural input lattice is converted to a transducer *I:G*, where *G* are gesture symbols (including *SEM*) and *I* contains both gesture symbols and the specific contents. *I* and *G* differ only in cases where the gesture symbol on *G* is *SEM* in which case the corresponding *I* symbol is the specific interpretation. After multimodal integration a projection *G:M* is taken from the result *G_W:M* machine and composed with the original *I:G* in order to reincorporate the specific contents that had to be left out of the finite-state process (*I:G o G:M = I:M*).

The multimodal finite-state transducers used at runtime are compiled from a declarative multimodal context-free grammar which captures the structure and interpretation of multimodal and unimodal commands, approximated where necessary using standard approximation techniques (Nederhof, 1997). This grammar captures not just multimodal integration patterns but also the parsing of speech and gesture, and the assignment of meaning. In Figure 9 we present a small fragment capable of handling MATCH commands such as *phone numbers for these three restaurants*. A multimodal CFG differs from a normal CFG in that the terminals are triples: *W:G:M*, where *W* is the speech stream (words), *G* the gesture stream (gesture symbols) and *M* the meaning stream (meaning symbols). An XML representation for meaning is used to facilate parsing and logging by other system components. The meaning tape symbols concatenate to form coherent XML expressions. The epsilon symbol (*eps*) indicates that a stream is empty in a given terminal.

Consider the example above where the user says *phone numbers for these three restaurants* and circles two groups of the restaurants (Figure 3). The gesture lattice Figure 8 is turned into a transducer *I:G* with the same symbol on each side except for the *SEM* arcs which are split. For example, path 15-16 *SEM([id1,id2,id3])* becomes *[id1,id2,id3]:SEM*. After *G* and the speech *W* are integrated using *G:W* and *G_W:M*. The *G* path in the result is used to re-establish the connection between *SEM* symbols and their specific contents in *I:G* (*I:G o G:M = I:M*). The meaning read off *I:M* is *<cmd> <phone> <restaurant> [id1,id2,id3] </restaurant> </phone> </cmd>*. This is passed to the multimodal dialog manager (MDM) and from there to the Multimodal UI where it results in the display in Figure 4 and coordinated TTS output. Since the speech input is a lattice and there is potential for ambiguity in the multimodal grammar, the output from MMFST to MDM is in fact a lattice of potential meaning representations.

S	→	eps:eps:<cmd> CMD eps:eps:</cmd>
CMD	→	phone:eps:<phone> numbers:eps:eps for:eps:eps DEICTICNP eps:eps:</phone>
DEICTICNP	→	DDETPL eps:area:eps eps:selection:eps NUM RESTPL eps:eps:<restaurant> eps:SEM:SEM eps:eps:</restaurant>
DDETPL	→	these:G:eps
RESTPL	→	restaurants:restaurant:eps
NUM	→	three:3:eps

Figure 9: Multimodal grammar fragment

Multimodal Dialog Manager (MDM) The MDM is based on previous work on speech-act based models of dialog (Stent et al., 1999; Rich and Sidner, 1998). It uses a Java-based toolkit for writing dialog managers that embodies an approach similar to that used in TrindiKit (Larsson et al., 1999). It includes several rule-based processes that operate on a shared state. The state includes system and user intentions and beliefs, a dialog history and focus space, and information about the speaker, the domain and the available modalities. The processes include an interpretation process, which selects the most likely interpretation of the user's input given the current state; an update process, which updates the state based on the selected interpretation; a selection process, which determines what the system's possible next moves are; and a generation process, which selects among the next moves and updates the system's model of the user's intentions as a result. MDM passes messages on to either the text planner or directly back to the Multimodal UI, depending on whether the selected next move represents a domain-level or communication-level goal.

In the route query example in Section 1, MDM first receives a route query in which only the destination is specified *How do I get to this place?*. In the selection phase it consults the domain model and determines that a source is also required for a route. It adds a request to query the user for the source to the system's next moves. This move is selected and the generation process selects a prompt and sends it to the TTS component. The system asks *Where do you want to go from?*. If the user says or writes *25th Street and 3rd Avenue* then MMFST will assign this input two possible interpretations. Either this is a request to zoom the display to the specified location or it is an assertion of a location. Since the MDM dialogue state indicates that it is waiting for an answer of the type location, MDM reranks the assertion as the most likely interpretation from the meaning lattice. A generalized overlay process (Alexandersson and Becker, 2001) is used to take the content of the assertion (a location) and add it into the partial route request. The result is determined to be complete and passed on to the UI, which resolves the location specifications to map coordinates and passes on a route request to the SUBWAY component.

Subway Route Constraint Solver (SUBWAY) This component has access to an exhaustive database of the NYC subway system. When it receives a route request with the desired source and destination points from the Multimodal UI, it explores the search space of possible routes in order to identify the optimal route, using a cost function based on the number of transfers, overall number of stops, and the distance to walk to/from the station at each end. It builds a list of the actions required to reach the destination and passes them to the multimodal generator.

Multimodal Generator The multimodal generator processes action lists from SUBWAY and other components and assigns appropriate prompts for each action. The result is a 'score' of prompts and actions which is passed to the Multimodal UI. The Multimodal UI plays this score by coordinating presentation of the graphical consequences of actions (Figure 6) with the corresponding TTS prompts.

Text-to-speech (TTS) AT&T's next-generation text-to-speech engine is integrated into the architecture and used to provide spoken output of restaurant information such as addresses and reviews, and for subway directions. The TTS agent provides progress notifications that are used by the Multimodal UI to coordinate speech with graphical displays.

Text Planner and User Model The text planner receives instructions from MDM for execution of 'compare', 'summarize', and 'recommend' commands. It uses a user model based on multi-attribute decision theory (Carenini and Moore, 2000; Carenini and Moore, 2001). For example, in order to make a comparison between the set of restaurants shown in Figure 5, the text planner first ranks the restaurants within the set according to the predicted ranking of the user model. Then, after selecting a small set of the highest ranked restaurants, it utilizes the user model to decide which restaurant attributes are important to mention. The resulting text plan is eventually converted to text and sent to TTS.

A user model for someone who cares most highly about cost and secondly about food quality and decor leads to a system response such as that in example Compare-A above. A user model for someone whose selections are driven by food quality and food type first, and cost only second, results in a system response such as that shown in Compare-B.

Compare-B: *Among the selected restaurants, the following offer exceptional overall value. Babbo's price is 60 dollars. It has superb food quality. Il Mulino's price is 65 dollars. It has superb food quality. Uguale's price is 33 dollars. It has excellent food quality.*

Note that the restaurants selected for the user who is not concerned about cost includes two rather more expensive restaurants that are not selected by the text planner for the cost-oriented user. Anonymous(2002) provides more detail on the text planner and user model.

Multimodal Logger In order to enable user studies, multimodal data collection, and debugging, the MATCH agents are instrumented so that they send details of user inputs, system outputs, and results of intermediate stages to a logger agent which records them in an XML log format that we have devised for multimodal interactions. Critically, we wanted to be able to collect data continually through system development and to do so not just in the lab but also in mobile settings. This meant we could not rely on always videotaping user interactions, but we still needed high fidelity playback of multimodal interaction. To overcome this problem, along with the user's ink we also log the current state of the UI and we have augmented the Multimodal UI so that it can dynamically replay user's speech and ink as they were received and show how the system responded. The browser and component-based nature of the Multimodal UI made it straightforward to reuse it to build a Log Viewer that can run over multimodal log files, replay interactions between the user and system, and allow analysis and annotation of the data. This system is related in function to STAMP (Oviatt and Clow, 1998) but does not require multimodal interactions to be videotaped. The ability of the system to run standalone is an important design feature since it enables testing and collection of multimodal data in realistic mobile environments without relying on the availability of a wireless network.

3 Experimental Evaluation

Our multimodal logging infrastructure has enabled MATCH to undergo continual user trials and experimental evaluation throughout its development. Repeated evaluations with small numbers of test users both in the lab and in mobile settings (Figure 10) have guided the design and iterative development of the system.

This iterative development approach highlighted several important problems early on. For example, while it was originally thought that users would formulate queries and navigation commands primarily by specifying the names of New York neighborhoods, as in *show Italian restaurants in Chelsea*, early field test studies in the city revealed that the need for neighborhood names in the grammar was minimal compared to the need for cross-streets and points of interest; hence, cross-streets and a sizable list of landmarks were added. Other early tests revealed the need for easily accessible 'cancel' and 'undo' features that allow users to make quick corrections. We also discovered that speech recognition performance was initially hindered by placement of

Figure 10: Testing MATCH in NYC

the 'click-to-speak' button and the recognition feedback box on the bottom-right side of the device, leading many users to speak 'to' this area, rather than toward the microphone on the upper left side. This placement also led left-handed users to block the microphone with their arms when they spoke. Moving the button and the feedback box to the top-left of the device overcame both of these problems.

After some initial open-ended piloting trials, more structured user tests were conducted, for which we developed a set of scenarios that required the test user to solve everyday problems using the system. These scenarios were left as open-ended as possible so the user would approach problems by whatever means came most naturally:

Sample scenario:*You have plans to meet your aunt for dinner later this evening at a Thai restaurant on the Upper West Side near her apartment on 95th St. and Broadway. Unfortunately, you forgot what time you're supposed to meet her, and you can't reach her by phone. Use MATCH to find the restaurant and write down the restaurant's telephone number so you can check on the reservation time.*

Test users were given minimal training, receiving a brief tutorial that was intentionally vague and broad in scope so the users might overestimate the system's capabilities and approach problems in new ways. Figure 11 summarizes results from our last scenario-based data collection for a fixed version of the system. There were five subjects (2 male, 3 female) none of whom had been involved in the development of the system. 'asr' and 'hw' stand for speech and handwriting recognition respectively.

Although ASR accuracy was low, overall task completion was high, suggesting that given the multimodal aspects of the system users were still able to complete tasks successfully. Unimodal pen com-

exchanges	338		asr word accuracy	59.6%
speech only	171	51%	asr sent. accuracy	36.1%
multimodal	93	28%	hw sent. accuracy	64%
pen only	66	19%	task completion rate	85%
GUI actions	8	2%	average time/task	6.25m

Figure 11: MATCH Performance

mands were recognized more successfully than spoken commands, however only 19% of commands are pen only. In ongoing work, we are exploring strategies to increase users' adoption of more robust pen-based input.

MATCH has a very fast system response time. Benchmarking a set of speech, pen, and multimodal commands, the average response time is approximately 3 seconds (time from end of user input to system response).

We are currently completing a larger scale scenario-based evaluation and an independent evaluation of the functionality of the text planner.

In addition to MATCH, the same multimodal architecture has been used for two other applications: a multimodal interface to corporate directory information and messaging and a medical application to assist emergency room doctors. The medical application is the most recent and demonstrates the utility of the architecture for rapid prototyping. System development took under two days for two people

4 Conclusion

The MATCH architecture enables users to interact flexibly using speech, pen, or synchronized combinations of the two depending on their preferences, task, and physical and social environment. The system responds by generating coordinated multimodal presentations adapted to the multimodal dialog context and user preferences. Features of the system such as the browser-based UI and the general purpose finite-state architecture for multimodal integration facilitate rapid prototyping and reuse of the technology for different applications. The multimodal logging infrastructure has enabled an iterative process of pro-active evaluation and data collection throughout system development. Since we can replay multimodal interactions without video we have been able to log and annotate subjects both in the lab and in NYC throughout the development process and use their input to drive system development.

References

J. Alexandersson and T. Becker. 2001. Overlay as the basic operation for discourse processing in a multimodal dialogue system. In *2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems*.

J. Allen, D. Byron, M. Dzikovska, G. Ferguson, L. Galescu, and A. Stent. 2000. An architecture for a generic dialogue shell. *Natural Language Engineering*, 6(3).

E. André. 2002. Natural language in multimedia/multimodal systems. In Ruslan Mitkov, editor, *Handbook of Computational Linguistics*. Oxford University Press.

Anonymous. 2002. An empirical evaluation of decision-theoretic methods for generating concise and user-tailored responses in the match multimodal dialogue system. In *In Submission*.

G. Carenini and J. D. Moore. 2000. An empirical study of the influence of argument conciseness on argument effectiveness. In *Proceedings of the 38th ACL*.

G. Carenini and J. D. Moore. 2001. An empirical study of the influence of user tailoring on evaluative argument effectiveness. In *IJCAI*, pages 1307–1314.

M. Johnston and S. Bangalore. 2000. Finite-state multimodal parsing and understanding. In *Proceedings of COLING 2000*, Saarbrücken, Germany.

M. Johnston and S. Bangalore. 2001. Finite-state methods for multimodal parsing and integration. In *ESSLLI Workshop on Finite-state Methods*, Helsinki, Finland.

M. Johnston. 2000. Deixis and conjunction in multimodal systems. In *Proceedings of COLING 2000*, Saarbrücken, Germany.

S. Larsson, P. Bohlin, J. Bos, and D. Traum. 1999. Trindikit manual. Technical report, TRINDI Deliverable D2.2.

D. Martin, A. Cheyer, and D. Moran. 1999. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1–2):91–128.

Mark-Jan Nederhof. 1997. Regular approximations of cfls: A grammatical view. In *Proceedings of the International Workshop on Parsing Technology*, Boston.

S. L. Oviatt and J. Clow. 1998. An automated tool for analysis of multimodal system performance. In *Proceedings of the International Conference on Spoken Language Processing*.

C. Rich and C. Sidner. 1998. COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3–4):315–350.

D. Rubine. 1991. Specifying gestures by example. *Computer graphics*, 25(4):329–337.

A. Stent, J. Dowding, J. Gawron, E. Bratt, and R. Moore. 1999. The CommandTalk spoken dialogue system. In *Proceedings of ACL'99*.