Proceedings of the 40th Annual Meeting of the Association for
Computational Linguistics (ACL), Philadelphia, July 2002, pp. 376-383.

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 07932, USA
johnston,srini,guna,ehlen,walker,stevew,pmaloor@research.att.com
*Now at SUNY Stonybrook, stent@cs.sunysb.edu

Abstract

Mobile interfaces need to allow the user and system to adapt their choice of communication modes according to user preferences, the task at hand, and the physical and social environment. We describe a multimodal application architecture which combines finite-state multimodal language processing, a speech-act based multimodal dialogue manager, dynamic multimodal output generation, and user-tailored text planning to enable 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.

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, 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 speech and graphics (See (André, 2002) for a detailed overview of previous work on multimodal input and output). Since mobile devices are used in different physical and social environments, for different tasks, by different 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 is most appropriate, and system output should be dynamically tailored so that it is maximally effective given the situation and the user's preferences.

We present our testbed multimodal application MATCH (Multimodal Access To City Help) and the general purpose multimodal architecture underlying it, that: is designed for highly mobile applications; enables flexible multimodal input; and provides flexible user-tailored multimodal output.

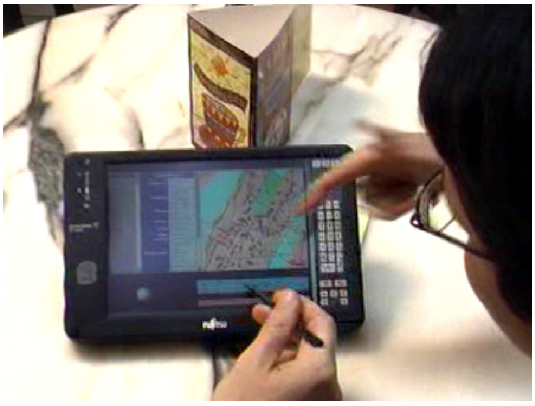

Figure 1: MATCH running on Fujitsu PDA

Highly mobile MATCH is a working city guide 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), and can also run in client-server mode across a wireless network.

Flexible multimodal input Users interact with a graphical interface displaying restaurant listings and a dynamic map showing locations and street information. They are free to provide input using speech, by drawing on the display with a stylus, or by using synchronous multimodal combinations of the two modes. For example, a user might ask to see cheap

Italian restaurants in Chelsea by saying *show cheap italian restaurants in chelsea*, by circling an area on the map and saying *show cheap italian restaurants in this neighborhood*; or, in a noisy or public environment, by circling an area and writing *cheap* and *italian* (Figure 2). The system will then zoom to the appropriate map location and show the locations of restaurants on the map. Users can ask for information about restaurants, such as phone numbers, addresses, and reviews. For example, a user might circle three restaurants as in Figure 3 and say *phone numbers for these three restaurants* (or write *phone*). Users can also manipulate the map interface directly. For example, a user might say *show upper west side* or circle an area and write *zoom*.

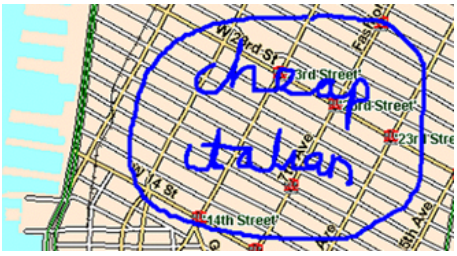

Figure 2: Unimodal pen command

Flexible multimodal output MATCH provides flexible, synchronized multimodal generation and can take initiative to engage in information-seeking subdialogues. If a user circles the three restaurants in Figure 3 and writes *phone*, the system responds with a graphical callout on the display, synchronized with a text-to-speech (TTS) prompt of the phone number, for each restaurant in turn (Figure 4).

Figure 3: Two area gestures

Figure 4: Phone query callouts

The system also provides subway directions. 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 of instructions. It 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 TTS prompts. Figure 5 shows the final display of a subway route heading downtown on the 6 train and transferring to the L train Brooklyn bound.

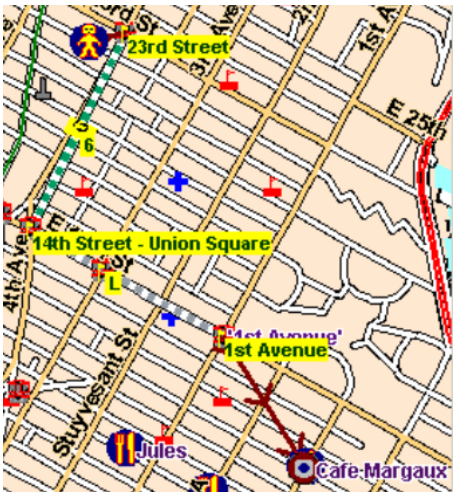

Figure 5: Multimodal subway route

User-tailored generation MATCH can also provide a user-tailored summary, comparison, or recommendation for an arbitrary set of restaurants, using a quantitative model of user preferences (Walker et al., 2002). The system will only discuss restaurants that rank highly according to the user's dining preferences, and will only describe attributes of those restaurants the user considers important. This permits concise, targeted system responses. For example, the user could say *compare these restaurants* and circle a large set of restaurants (Figure 6). If the user considers inexpensiveness and food quality to be the most important attributes of a restaurant, the system response might be:

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 6: Comparing a large set of restaurants

2 Multimodal Application Architecture

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

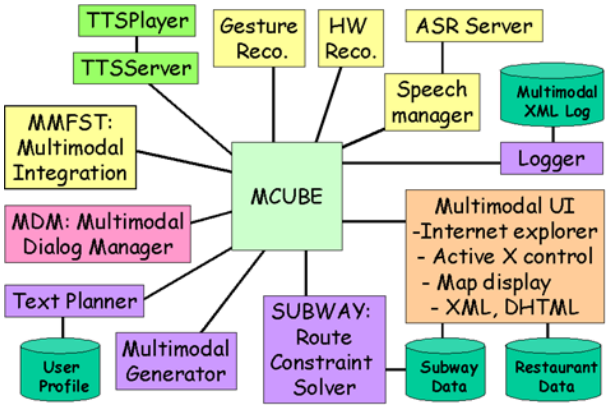

Figure 7: Multimodal Architecture

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), the use of KQML for messaging in Allen et al (2000), and the Communicator hub (Seneff et al., 1998). 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, providing a general mechanism for message parsing and facilitating logging.

Multimodal User Interface 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.

MATCH uses a control that provides a dynamic pan-able, zoomable map display. The control has ink handling capability. This enables 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, are identified. The electronic ink is broken into a lattice of strokes and sent to the gesture recognition and handwriting recognition components which enrich this stroke lattice with possible classifications of strokes and stroke combinations. The UI then translates this stroke lattice into an ink meaning lattice representing all of the possible interpretations of the user's ink and sends it to MMFST.

In order to provide spoken input the user must tap a click-to-speak button on the Multimodal UI. We found that in an application such as MATCH which provides extensive unimodal pen-based interaction, it is 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.

The Multimodal UI also provides graphical output capabilities and performs synchronization of multimodal output. For example, it synchronizes the display actions and TTS prompts in the answer to the route query mentioned in Section 1.

Speech Recognition MATCH uses AT&T's Watson speech recognition engine. A speech manager running on the device gathers audio and communicates with a recognition server running either on the device or on the network. The recognition server provides word lattice output which is passed to MMFST.

Gesture and handwriting recognition Gesture and handwriting recognition agents provide possible classifications of electronic ink for the UI. Recognitions are performed both on individual strokes and combinations of strokes in the input ink lattice. 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 classi-

fying 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 provide a rich visual vocabulary for multimodal and pen-based commands.

Gestures are represented in the ink meaning lattice 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.

When multiple selection gestures are present an aggregation technique (Johnston and Bangalore, 2001) is employed to overcome the problems with deictic plurals and numerals described in Johnston (2000). Aggregation augments the ink meaning 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 ink meaning 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 ink meaning lattice (from the UI) and builds a multimodal meaning lattice which captures the potential joint interpretations of the speech and ink inputs. MMFST is able to provide rapid response times by making unimodal timeouts 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 ink meaning lattice, otherwise it applies a short timeout (1 sec.) and treats the speech as unimodal. When an ink meaning lattice arrives, if the user has tapped click-to-speak MMFST waits for the speech lattice to arrive, otherwise it applies a short timeout (1 sec.) and treats the ink as unimodal.

MMFST uses the finite-state approach to multimodal integration and understanding proposed by Johnston and Bangalore (2000). Possibilities for multimodal integration and understanding are captured in a three tape device in which the first tape represents the speech stream (words), the second the ink stream (gesture symbols) and the third their combined meaning (meaning symbols). In essence, this device takes the speech and ink meaning lattices as inputs, consumes them using the first two tapes, and writes out a multimodal meaning lattice using the third tape. The three tape finite-state device is simulated using two transducers: *G:W* which is used to align speech and ink and *G_W:M* which takes a composite alphabet of speech and gesture symbols as input and outputs meaning. The ink meaning 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 possible sequences of coordinates that could occur in an area gesture cannot be encoded in the finite-state device. We employ the approach proposed in (Johnston and Bangalore, 2001) in which the ink meaning 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 were 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

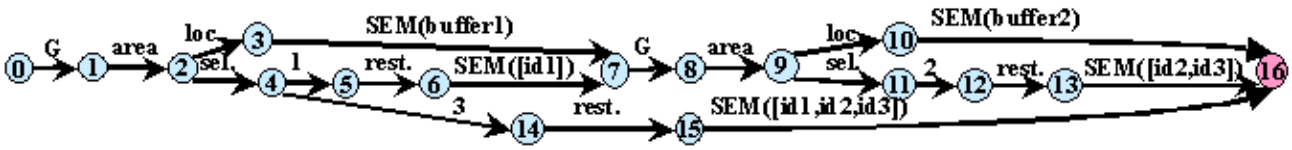

Figure 8: Ink Meaning Lattice

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 simplified 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 ink 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.

When the user says *phone numbers for these three restaurants* and circles two groups of 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 resulting in a display like Figure 4 with coordinated TTS output. Since the speech input is a lattice and there is also potential for ambiguity in the multimodal grammar, the output from MMFST to MDM is an N-best list of potential multimodal interpretations.

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 is similar in philosophy to TrindiKit (Larsson et al., 1999). It includes several rule-based

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

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 interpretation, update, selection and generation processes.

The interpretation process takes as input an N-best list of possible multimodal interpretations for a user input from MMFST. It rescores them according to a set of rules that encode the most likely next speech act given the current dialogue context, and picks the most likely interpretation from the result. The update process updates the dialogue context according to the system's interpretation of user input. It augments the dialogue history, focus space, models of user and system beliefs, and model of user intentions. It also alters the list of current modalities to reflect those most recently used by the user.

The selection process determines the system's next move(s). In the case of a command, request or question, it first checks that the input is fully specified (using the domain ontology, which contains information about required and optional roles for different types of actions); if it is not, then the system's next move is to take the initiative and start an information-gathering subdialogue. If the input is fully specified, the system's next move is to perform the command or answer the question; to do this, MDM communicates with the UI. Since MDM is aware of the current set of preferred modalities, it can provide feedback and responses tailored to the user's modality preferences.

The generation process performs template-based generation for simple responses and updates the system's model of the user's intentions after generation. The text planner is used for more complex genera-

tion, such as the generation of comparisons.

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. 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. The UI resolves the location to map coordinates and passes on a route request to the SUBWAY component.

We found this traditional speech-act based dialogue manager worked well for our multimodal interface. Critical in this was our use of a common semantic representation across spoken, gestured, and multimodal commands. The majority of the dialogue rules operate in a mode-independent fashion, giving users flexibility in the mode they choose to advance the dialogue. On the other hand, mode sensitivity is also important since user modality choice can be used to determine system mode choice for confirmation and other responses.

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 to identify the optimal one, using a cost function based on the number of transfers, overall number of stops, and the walking distance from the station at each end. It builds a list of actions required to reach the destination and passes them to the multimodal generator.

Multimodal Generator and Text-to-speech The multimodal generator processes action lists from SUBWAY and other components and assigns appropriate prompts for each action using a template-based generator. The result is a 'score' of prompts and actions which is passed to the Multimodal UI. The Multimodal UI plays this 'score' by coordinating changes in the interface with the corresponding TTS prompts. AT&T's Natural Voices TTS engine is used to provide the spoken output. When the UI receives a multimodal score, it builds a stack of graphical actions such as zooming the display to a particular location or putting up a graphical callout. It then sends the prompts to be rendered by the TTS server. As each prompt is synthesized the TTS server sends progress notifications to the Multimodal UI, which pops the next graphical action off the stack and executes it.

Text Planner and User Model The text planner receives instructions from MDM for execution of 'compare', 'summarize', and 'recommend' commands. It employs a user model based on multi-attribute decision theory (Carenini and Moore, 2001). For example, in order to make a comparison between the set of restaurants shown in Figure 6, 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 converted to text and sent to TTS (Walker et al., 2002). 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 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.*

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.

Multimodal Logger User studies, multimodal data collection, and debugging were accomplished by instrumenting MATCH agents to send details of user inputs, system processes, and system outputs to a logger agent that maintains an XML log designed for multimodal interactions. Our critical objective was to collect data continually throughout system development, and to be able to do so in mobile settings. While this rendered the common practice of videotaping user interactions impractical, we still required high fidelity records of each multimodal interaction. To address this problem, MATCH logs the state of the UI and the user's ink, along with detailed data

from other components. These components can in turn dynamically replay the user's speech and ink as they were originally received, and show how the system responded. The browser- and component-based architecture of the Multimodal UI facilitated its reuse in a Log Viewer that reads multimodal log files, replays interactions between the user and system, and allows analysis and annotation of the data. MATCH's logging system is similar in function to STAMP (Oviatt and Clow, 1998), but does not require multimodal interactions to be videotaped and allows rapid reconfiguration for different annotation tasks since it is browser-based. The ability of the system to log data standalone is important, since it enables testing and collection of multimodal data in realistic mobile environments without relying on external equipment.

3 Experimental Evaluation

Our multimodal logging infrastructure enabled MATCH to undergo continual user trials and evaluation throughout 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.

Figure 10: Testing MATCH in NYC

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 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 resolved both of these problems.

After initial open-ended piloting trials, more structured user tests were conducted, for which we developed a set of six scenarios ordered by increasing level of difficulty. These required the test user to solve problems using the system. These scenarios were left as open-ended as possible to elicit natural responses.

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 received 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 system development. All of these five tests were conducted indoors in offices.

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

Figure 11: MATCH study

There were an average of 12.75 multimodal exchanges (pairs of user input and system response) per scenario. The overall time per scenario varied from 1.5 to to 15 minutes. The longer completion times resulted from poor ASR performance for some of the users. Although ASR accuracy was low, overall task completion was high, suggesting that the multimodal aspects of the system helped users to complete tasks. Unimodal pen commands were recognized more successfully than spoken commands; however, only 19% of commands were pen only. In ongoing work, we are exploring strategies to increase users' adoption of more robust pen-based and multimodal 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 prototype 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 rapid development of mobile multimodal applications. Combining finite-state multimodal integration with a speech-act based dialogue manager 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 general purpose finite-state architecture for multimodal integration facilitate rapid prototyping and reuse of the technology for different applications. The lattice-based finite-state approach to multimodal understanding enables both multimodal integration and dialogue context to compensate for recognition errors. 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.

Acknowledgements

Thanks to AT&T Labs and DARPA (contract MDA972-99-3-0003) for financial support. We would also like to thank Noemie Elhadad, Candace Kamm, Elliot Pinson, Mazin Rahim, Owen Rambow, and Nika Smith.

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. *JNLE*, 6(3).

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

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.

M-J. 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 ICSLP*.

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.

S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue. 1998. Galaxy-II: A reference architecture for conversational system development. In *ICSLP-98*.

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

M. A. Walker, S. J. Whittaker, P. Maloor, J. D. Moore, M. Johnston, and G. Vasireddy. 2002. Speech-Plans: Generating evaluative responses in spoken dialogue. In *In Proceedings of INLG-02*.