CREATING MULTI-MODAL, USER-CENTRIC RECORDS OF MEETINGS WITH THE CARNEGIE MELLON MEETING RECORDER ARCHITECTURE

Satanjeev Banerjee, Jason Cohen, Thomas Quisel, Arthur Chan, Yash Patodia,
Ziad Al Bawab, Rong Zhang, Alan Black, Richard Stern, Roni Rosenfeld,
Alexander I. Rudnicky, Paul Rybski, Manuela Veloso
{banerjee+,air}@cs.cmu.edu

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA

ABSTRACT

Our goal is to build conversational agents that combine information from speech, gesture, hand-writing, text and presentations to create an understanding of the ongoing conversation (e.g. by identifying the action items agreed upon), and that can make useful contributions to the meeting based on such an understanding (e.g. by confirming the details of the action items).

To create a corpus of relevant data, we have implemented the Carnegie Mellon Meeting Recorder to capture detailed multi-modal recordings of meetings. This software differs somewhat from other meeting room architectures in that it focuses on instrumenting the individual rather than the room and assumes that the meeting space is not fixed in advance. Thus, most of the sensors are user-centric (close-talking microphones connected to laptop computers, instrumented note-pads, instrumented presentation software, etc), although some are indeed "room-centric" (instrumented whiteboard, distant cameras, table-top microphones, etc).

This paper describes the details of our data collection environment. We report on the current status of our data collection, transcription and higher-level discourse annotation efforts. We also describe some of our initial research on conversational turn-taking based on this corpus.

1. INTRODUCTION

Our long–term goal is to build agents that can assist professionals in their day-to-day work life. In particular, we are interested in building *conversational* agents that can efficiently and effectively participate in multi-party meetings and help meeting participants with meeting-related tasks. Such tasks may include tracking the agenda items being covered, noting action items and milestones being decided upon, and once the meeting is done, creating a summary of the highlights of the meeting.

Such agents, CALOs (Cognitive Agent that Learns and Organizes),[1] can conceivably be of two types: a *personal* CALO that focuses on an individual meeting participant, processing their speech and gestures, interactions with their computer, etc, and a *public* CALO that focuses on the meeting itself, processing the combination of the various speech and gestures, the mutual gaze information of participants, the strokes on a white board, etc. Thus a single meeting can have multiple such CALOs that may cooperate with each other and with the participants themselves.

To study how human beings behave in a meeting, and to create a corpus of meeting data, we are collecting detailed multi-modal records of actual meetings held at Carnegie Mellon University. Such meetings have a wide range in terms of the number of participants (from two to a dozen), the length of the meeting (15 mins to two hours), the content of the discussion (core speech group meetings, spoken dialog group meetings, vision group meetings, etc), and the goals of the meeting (student-advisor, weekly report meetings, scheduling meetings, brain-storming sessions, etc). At each such meeting, we record different information streams that can be classified as personal or public streams. The aim is to collect the kinds of streams that will be used by personal and public CALOs. The personal data streams include speech from a close-talking microphone, notes typed on the participant's notebook, and information from presentation slides. Public data streams include speech from table-top microphones, video collected using the panoramic CAMEO system, video collected using a single whole-scene camera, and strokes drawn on a whiteboard.

These various streams of information need to be captured, time synchronized with respect to each other and collected in a central repository. In this paper we present the architecture we are developing for data capture, and describe

[1]For more information about the CALO Project, please refer to http://www.ai.sri.com/project/CALO.

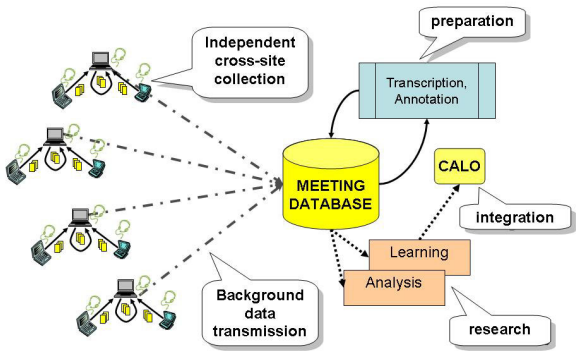

Fig. 1. *Meeting Recorder System Architecture.*

the Carnegie Mellon Meeting Recorder system that implements the architecture. We also describe our current data collection status, and finally report on some initial results from the analysis of the collected data.

2. THE MEETING RECORDER ARCHITECTURE

Over time we expect to incorporate additional sensors to create more detailed records of meetings. Keeping this in mind, our goal is to create a recording architecture that allows rapid, simple incorporation of new information streams. In this section we describe the meeting recorder architecture from the point of view of how a stream of information is incorporated into the meeting record.

2.1. An Event View of Information Streams

An information stream can consist of either a sequence of discrete events (such as the utterances from a speech stream) or one long continuous event whose beginning and end coincide with the beginning and end of the meeting (such as a video stream). Further, discrete events can be instantaneous (such as pressing a key on the keyboard) or may involve a variable amount of time. We treat all three of these streams in the same way, and expect each stream to generate events that have a start time and an end time. Thus continuous streams generate only one event, while for instantaneous events, the start and the end time are the same and coincide with when the event took place.

2.2. Synchronizing Time Stamps

Each generated event must be associated with a start and end time stamp. It is crucial that these time stamps be *synchronized* with those generated for events on other streams. That is, if two events on two different streams start simultaneously, the generated start time stamps should also be the same. To perform such synchronization we have implemented a light-weight version of the Simplified Network Time Protocol (SNTP). This module queries the time on a reference NTP server and also estimates the round-trip time of the query. Using this information, the module estimates the difference in clock time between the local machine (on which the information stream is being captured) and the NTP server. The stream capturing module can then use this information to put time stamps on the events that reflect the time on the NTP reference server. By synchronizing every stream capturing machine to a single NTP server, all the streams can be synchronized.

To empirically establish the synchronization accuracy this method achieves on our network we collected reference clock time stamps and found the variance in these times to be under 10ms. This implementation is sub-optimal since it assumes that the network is high-speed, which may not always be the case. As future work we plan to implement more sophisticated synchronization techniques. Note that not all data streams are capable of equal resolution. For example, a conventional video stream has an inherent lower limit tied to its frame rate (30 fps). Nevertheless other phenomena, such as managing turn-taking may require fairly high temporal resolution.

2.3. Identifying Information for Events

Generated events include other information that helps identify the event. These include identifiers for the meeting, the name of the participant, whether the information stream is user-focused, and the modality (speech/video/hand-writing etc).

2.4. Data Aggregation

As several different CALO sites will be making use of the meeting corpus, we have developed a process for aggregating data on a central server. Data aggregation is a two-step process. During the meeting all data are stored locally. This preserves bandwidth and allows flexibility in the physical location of the meeting (in the extreme, no external connections would be necessary). After the end of the meeting and subject to bandwidth availability, all data are uploaded to the server, as a background process.

The Microsoft Background Intelligent Transfer Service (BITS) protocol is used to manage uploading. We implemented a BITS client, termed wxBITS, that runs as a separate background process and transparently uploads event files. A stream capturing system saves event data files to the local machine, and spawns the wxBITS client if it is not already running. The wxBITS client checks the event data directory, and uploads any data file therein using the BITS

protocol to a Microsoft IIS BITS-enabled web server, and then deletes the files from the local machine. While uploading, the client can throttle its use of the network bandwidth if it detects high network activity on the local machine so that the act of uploading does not obstruct other work on the local machine. Further, if the connection to the server is lost during upload, the wxBITS client has the capability of restarting the upload from the point at which it had stopped. This robustness ensures that the event data from a meeting will reach the server over a reasonably small interval. If wxBITS detects that it has not uploaded any files in 60 seconds, it self terminates to save system resources.

2.5. Logging Event Information

Although event data is not uploaded to the server as soon as the event is generated, we do send a *summary block* with event information (time stamps, identifying information, etc) to a central server. While not strictly necessary for creating a record of the meeting, this creates a redundant record of session events that can be used for debugging purposes. Presently, event summaries can be sent to a host running a PERL script that collates the information into an HTML file, or to a Timeline Server through the Open Agent Architecture (OAA). The format of the event summary allows for new streams to be easily incorporated without the need to alter the underlying protocol.

We have used the above architecture to implement the capture, synchronization and storage of several information streams. In the following sections we describe the characteristics of each stream capture component.

3. CAPTURING CLOSE-TALKING SPEECH AND TYPED NOTES

Our principal interest is the speech produced by meeting participants. In our meetings, each participant wears a head-mounted microphone (typically an Emkay VR-3345 microphone, though individuals may show up with a different though similar headset on occasion). To record the speech from these microphones, as well as to capture the notes typed out by the participant on their notebook, we have implemented the Meeting Recorder Cross Platform (MRCP), a screen shot of which is shown in figure 2. To accommodate personal choices of computer, we have implemented MRCP in a platform independent manner, usable on Windows, Linux and Macintosh OSX systems.

3.1. Processing the Speech Stream

The MRCP implementation borrows components of the CMU Sphinx 2 real-time recognition system, such as end-pointing and utterance management (this was done with a view to later integration of recognition capabilities into the personal

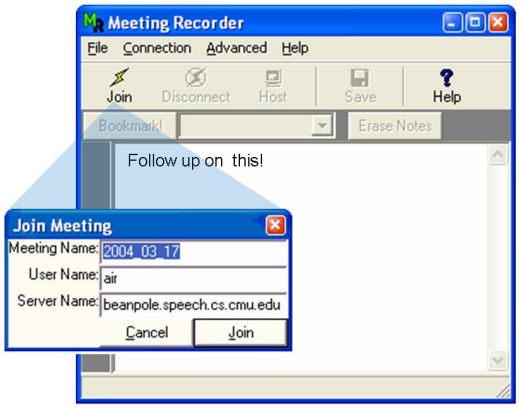

Fig. 2. *Meeting Recorder Client.*

CALO). For recording, we have standardized on a 11.025 kHz sampling rate. This sampling rate, although not widely used in the ASR community, provides maximum flexibility in choice of platform. It does not result in significant degradation of recognition relative to a 16kHz sampling rate. Recording stops when the end of speech is detected; the resulting utterance constitutes an event and is associated with synchronized time stamps (obtained with the help of the NTP module described above) and saved to local disc. A summary of this event is sent to a central server, and the utterance file itself is queued up for transfer to the central storage using wxBITS.

To make MRCP portable, the audio core of Sphinx 2 was updated to use PortAudio,[2] a cross-platform low-level audio API. Multi-threading allows MRCP to manage GUI interaction, disk usage, network usage and other computation. A high priority thread receives audio and perform silence removal. One or more separate threads retrieve the audio in the form of buffered utterances and perform other, potentially time-consuming operations, such as sending network messages, storing audio to disk, speech processing, etc.

Audio input must be available with low latency for real-time level monitoring and interactive speech control. The audio must also be available to higher latency threads, such as writing entire utterances to disk. To make both situations simple, we created a structure to buffer utterances separately along with information about each utterance. The structure representing an utterance contains a FIFO queue for audio, so utterance length is not limited by memory, only disk space. Utterances themselves are also queued, allowing slow threads to lag many utterances behind.

[2]PortAudio is a free, cross platform, open-source, audio I/O library. Information is available at `http://www.portaudio.com/`

3.2. Processing the Note-Taking Stream

The MRCP interface provides a text area within which participants can take notes during the meeting. This text area mirrors Windows Notepad in functionality; as we gain experience with MRCP, we plan to add to the capabilities of this interface.

While the decision of how to discretize the speech stream into events is relatively straightforward, it is not equally clear what constitutes an event in the notes stream. One possibility is to treat the stream as being a continuous one, generating one note taking "event" that starts with a blank slate at the start of the meeting, and ends with a full set of notes at the end. However this view does not allow us to capture detailed timing information for the various segments (say sentences) in the notes. The other extreme possibility is to generate an instantaneous event for each key press; this approach is not satisfying because it splits the notes into unintelligible pieces making it harder to work with the events. As a middle ground we generate an event every time the Enter key is pressed. The data for each event consists of all the text in the area, thereby creating a record of everything that has changed since the last press of the Enter key. We anticipate that this information will be useful as a source of evidence for higher-level processing of the meeting, for example topic-spotting and segmentation.

As before, these event files are time stamped, their summaries are sent to the server, and the files are queued up for transfer using wxBITS.

3.3. Interface and Networking Issues with MRCP

To support platform independence, the MRCP is based on wxWidgets (formerly known as wxWindows[3]) for its graphical user interface (GUI), file access, and networking capabilities. The MRCP interface has been designed to be fast, modern, and easy to use. Users begin recording speech and notes by "Join"-ing at which stage they enter the name of the meeting which is used as a way to group together all the event files generated by the various participants during a single meeting. Besides the notes text box discussed above, the interface also features a toolbar, a volume meter, and a status bar. The toolbar allows users to join a meeting with two clicks, disconnect with one, as well as save notes to a local file. The status bar reports useful status information, such as whether a meeting is joined and how many utterances have occurred. In a standard drop down menu bar, various menu items allow users to customize the Meeting Recorder's functionality. All customizations are stored in a settings file, which allows information such as default servers and usernames to be reused. The Meeting Recorder's user interface is controlled by an explicit finite state machine, which eliminates the possibility of bad user input and invalid state transitions by disabling and enabling controls at the appropriate times.

[3]wxWidgets provides a single API for writing GUI applications on multiple platforms. See `http://www.wxwidgets.org/` for more information

4. CAPTURING WHITEBOARD PEN STROKES

We use a Mimio device (www.mimio.com) to capture the pen strokes made on the whiteboard during a meeting. The device attaches to a normal whiteboard and uses infrared sensors to track marker position; data is transmitted to a computer using a USB cable. The commercial software that ships with the device allows users to save the contents of the whiteboard for later perusal. Our goal was to capture the pen strokes and draw temporal relations between them and other modalities, similar to the notes stream.

To do so, we have implemented the Mimio Logger that uses the Mimio Recorder API (along with sample code provided by the Oregon Health and Science University) to detect the state of the whiteboard (such as "pen down", "pen up", etc), and, when the pen is down, the sequence of X-Y coordinates that represents the trajectory that the pen is following on the board at that instant. We defined an event as the collection of the X-Y coordinates captured between pen down and pen up. As soon as a pen down is detected, the Mimio Logger starts recording the X-Y coordinates to a file. Recording stops when pen up is detected. The resulting file containing the X-Y coordinates represents a single stroke on the whiteboard, and constitutes a single event for this stream. As before, this event is time stamped using the NTP module, a summary block for this event is sent to the central server, and the file itself is queued up for transfer to the central storage using wxBITS.

When the Mimio Logger is started, the user is presented with a login screen where the user can enter the meeting name, user id, and the server to which the data must be uploaded.

5. CAPTURING SLIDE PRESENTATIONS

Often in the meetings we are recording, one or more participants makes a slide presentation. Our goal is to capture the information on each slide of the presentation, as well as capture the timing information of the slide changes, mouse button clicks, etc.

To do so, we have implemented the PowerPoint Scraper which uses the PowerPoint API provided by Microsoft to query the PowerPoint application for information such as the format of the slides being shown, their contents, etc. The PowerPoint Scraper queries PowerPoint at short intervals so that whenever a slide change occurs, the program obtains all the information on the new slide. Since this information is available instantaneously, events are instantaneous—that

is, the start time and the end time are both the same. The PowerPoint scraper creates an event every time the slide changes, and the data for this event consists of all the information on the new slide. As usual, these events are time stamped using the NTP module, summary blocks are sent to the central server, and the files themselves are eventually uploaded to a central storage using wxBITS.

6. CAPTURING VIDEO USING CAMEO

The Camera Assisted Meeting Event Observer (CAMEO)[4] has been developed to support research in the physical awareness aspect of the CALO personalized learning assistant. CAMEO is designed to be an inexpensive high-resolution omnidirectional vision system that lets us focus our research on multi-person meeting scenarios. Panoramic images are generated from CAMEO's circular camera configuration (see Figure 3). CAMEO was developed to help explore the issues involved with using a small portable sensor system for meeting understanding, rather than specially instrumenting a room for that task. Ultimately, CAMEO is a designed to be taken into a meeting environment and simply placed in the center of the room without requiring any special calibration.

There are a number of specific scientific issues that we are using CAMEO to address. The first is how to generate coherent high-resolution images in real time from multiple cameras. We are exploring automatic techniques by which the images are registered and merged by having CAMEO identify local feature points and correlating them between the images. Another area of research includes real-time person tracking within the generated panoramic images. We are examining methods that make use of sparse data, including local feature and color histogram information, in order to achieve rapid tracking performance. Face recognition algorithms are also being pursued as an independent area of research. A method has been developed by which CAMEO can extract faces from the image in real time, learn a classification dataset, and then use that dataset to classify the detected faces. Finally, we are trying to understand the scientific issues involved with inferring the state of individual people (their actions and activities) as well as the global state of the meeting (people's interactions) from only seeing the positions of people's faces. Because we make the assumptions of using CAMEO in a completely uncalibrated environment, CAMEO will only realistically be able to detect people when it sees their faces. This is currently an off-line process that operates after CAMEO has identified and tracked the positions of people in the video stream.

[4]Paul E. Rybski, Fernando de la Torre, Raju Patil, Carlos Vallespi, Manuela Veloso and Brett Browning, *CAMEO: Camera Assisted Meeting Event Observer*, In Proceedings of the IEEE International Conference on Robotics and Automation, 2004.

Fig. 3. *CAMEO panoramic video capture device.*

As part of the meeting recording architecture, CAMEO generates panoramic images in real time and streams them to an MPEG-4 movie file that can be used for off-line processing. While CAMEO can process video data directly from its cameras, it can load and process these movies off-line which allows the state information to be extracted and used as part of the larger meeting understanding problem.

7. CURRENT STATUS OF DATA COLLECTION

At this time we have collected a total of 14 meetings, containing a total of 11,559 utterances. Utterances in this case have been automatically segmented and this count should be understood as an overestimate, since some proportion of utterances will be non-speech vocalizations (such as loud breathing) or other sounds. On the other hand, when a presentation is being made, very long utterances will be recorded. A more accurate characterization of the evolving corpus will be available once transcription is completed.

8. INITIAL RESEARCH USING THE DATA

To participate in a meeting, an agent needs to be able to track the conversation in progress and create an understanding in terms of what topic is being discussed, what agenda points are being covered, what milestones are being decided upon, etc. A possible first step towards acquiring such a

set of capabilities is to be able to create a coarse judgment on the *state* of the meeting. Possible states include *discussion*, *information flow* (when one participant is giving information to one or more other participants), and *presentation* (when one participant is making a presentation to the whole group). Our initial goal is to see how well we can make such a classification using very simple speech–only information. (Later we plan to experiment with how much better we can do when we combine information from multiple modalities such as both speech and vision).

Towards this goal, we have hand-annotated about half an hour of a meeting with 5 participants. The annotation consists of three layers. At the highest layer, the annotator classified each point of the meeting as being in one of the three states above. At the next layer, the annotator specified the role of each participant during a particular meeting state. Roles include *presenter*, *information producer*, *information consumer*, etc. At the lowest level, the annotator precisely defined the start and stop times of each region of overlapped speech, along with who were speaking at all points of the meeting. We have found that it is easiest to do such annotation by observing the video captured by the single whole–scene camera, since that stream contains both audio and visual information that the human being can rely on to make the classifications.

Using this annotated data we trained a decision tree classifier to label each second of the meeting into one of the three meeting states, based on very simple speech features such as the number of speakers, the number of overlaps, the lengths of the utterances, etc. All such features are defined for a window immediately preceding the particular second of the meeting that is being classified.

We evaluated the learned models using a separate 15 min test set from a similar meeting and found that our simple model labeled meeting state with 51% accuracy and participant role at 53%, both at a window size of about 20 sec. The accuracy of both classifiers was well above chance.

Given the small amount of training data and the sparse feature set, we believe that much higher accuracies should be possible to achieve. Once available the resulting labelling can be used as a component of meeting understanding.

9. CONCLUSION

We have described an architecture for flexible multi-stream collection of meeting data. The architecture is still evolving but so far has shown itself to be a viable approach of meeting data collection, particularly with respect to our goal of creating a mobile and easy-to-use system that can be managed by a broad base of users.

10. ACKNOWLEDGEMENT

This work was supported by DARPA grant NBCH-D-03-0010. The content of the information in this publication does not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred.