Multimodal Play Back of Collaborative Multiparty Corpora

Edward C Kaiser[1]
Oregon Health and Science University.
OGI School of Science & Eng .
20000 NW Walker Road
Beaverton, OR 97006, USA
+1 503 748 7803
kaiser@csee.ogi.edu

Paulo Barthelmess[1]
Oregon Health and Science University.
OGI School of Science & Eng .
20000 NW Walker Road
Beaverton, OR 97006, USA
+1 503 748 7803
paulo@cse.ogi.edu

Alexander Arthur[1]
Oregon Health and Science University.
OGI School of Science & Eng .
20000 NW Walker Road
Beaverton, OR 97006, USA
+1 503 748 7803
arthur@cse.ogi.edu

ABSTRACT

A basic capability driving the development of systems is the ability to execute multiple versions of a system against the same set of data. That is essential for instance to verify that coding problems were corrected, to provide guarantees that changes to the code did not introduce errors (regression testing) and for comparison of performance among different algorithmic solutions.

Multimodal systems process a combination of inputs and their results are susceptible to timing issues, which determine for instance whether a pair of inputs from different modalities combine or not. When a multimodal system is executed against corpus data, it becomes necessary to synchronize processing of the various input streams and to handle time-related information so as to emulate real-time execution. This is particularly complex when multiple components of the various input stream processors take longer than real-time. In this paper we describe a multimodal play back mechanism that addresses these problems.

We report on two data collection and play back software and hardware environments — one used for system development and evaluation and the other used for supporting manual annotation of multimodal data.

Categories and Subject Descriptors

H.5.3 [Group and Organization Interfaces]: Collaborative Computing; Synchronous interaction. H.5.2 [User Interfaces]: Natural language; Input devices and strategies. I.2.10 [Vision and Scene Understanding]: 3D/stereo scene analysis. I.2.6 [Learning]: Language Acquisition.

General Terms

Collaboration; Multimodal; Context-aware.

Keywords

Collaborative Interaction; Multimodal Corpora processing; Intelligent interfaces; Gesture; Vocabulary learning.

1. INTRODUCTION

Multimodal corpora processing introduces interesting problems related to timing and synchronization of the multiple components. When exploiting corpus data to drive system execution, care needs to be exercised to replicate closely enough the conditions that existed at the time the data was captured, so that the results of replaying do not deviate in a significant way from what system execution would have been if the application were executed in real-time. In a complex multimodal system the multiple data streams from different modalities all need to be replayed within temporal, semantic and other constraints. Play back becomes problematic when some components are not able to perform in real-time. This is a common situation for early versions of research components, for which performance issues have not yet been addressed. One would want in this case for the result of processing to be equivalent to that of a hypothetical real-time version of the system with equivalent functionality.

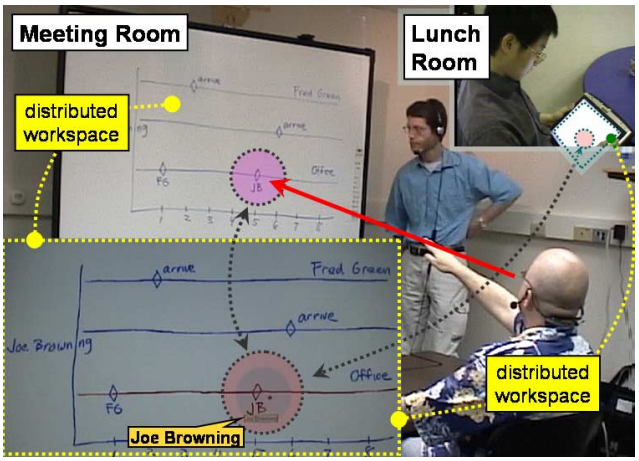

Figure 1. Charter System in use during a distributed 3-person scheduling meeting, with speech, sketch, handwriting and 3D gesture recognition. A semantic rendering of the schedule chart, the pointing reference (small red and large blue dots), and the dynamically discovered abbreviation semantics (*JB=Joe Browning*) are available to the remote participant on tablet PC.

In this paper we describe our experience designing and implementing mechanisms that provide play back capabilities to a complex sub-system of the CALO Project (Cognitive Assistant the Learns and Organizes) [1]. The Charter System provides assistance for small groups of people collaborating in the creation of project schedules (Fig. 1). We elaborate on the play back support issues that surround the multimodal integration and describe the functionality that was built to support multimodal debugging (Section 2). Previous versions of the system focused on the real-time processing of multimodal data for demonstration purposes. We have introduced functionality, overviewed in this

[1] Early work on this paper was conducted while the authors were at OGI. Later work was done while the authors were at Natural Interaction Systems, LLC. (http://www.naturalinteraction.com)

ICMI '05, October 4-6, 2005, Trento, Italy.
Workshop on Multimodal, Multiparty Meeting Processing

paper, to allow for the system to be driven by low-level, pre-recorded data. We then show two examples in which replay issues are of fundamental importance:

1) the Task Discussion system, into which Charter is embedded (Section 3) and
2) the data collection environment at the Oregon Graduate Institute (Section 4).

These two systems illustrate two uses of the play back functionality: supporting system evaluation, and supporting manual annotation of multimodal data respectively. The paper ends with some discussion of Related Work (Section 5), Conclusions and Future Work (Section 6) and references.

2. MULTIMODAL PLAY BACK AND DEBUGGING IN CHARTER

2.1 Overview

Charter assists small groups collaborating to design sketched project schedules based on a Gantt chart formalism. Ink captured from the instrumented whiteboard is recognized as graphical symbols and handwritten labels [2]. Redundant speech is used to disambiguate handwritten labels and to recover the semantics of abbreviations [3]. Pointing gestures towards the whiteboard are recognized within Charter, and the diagram elements likely to be the targets of the pointing are identified. Charter supports distributed collaboration both by allowing parts of a diagram to be sketched by participants at remote sites, and also by allowing gestures towards the whiteboard to be propagated among sites to promote awareness of remote pointing events [4]. Figure 1 shows an example of this collaboration and the services provided by Charter.

2.2 Architecture

The system is organized around a multiagent architecture, and is a direct descendant of the QuickSet [5] multimodal system and MAVEN [6], from which gesture recognition and multimodal integration components were taken. Figure 2 presents a (simplified) view of the system's architecture, including elements of the play back architecture. The input generated by the pen strokes on an instrumented board, and the speech and video captured from microphones and cameras are processed by components serving each of the potentially multiple sites involved in a collaboration.

The Charter system is comprised of recognizers for (cf. Fig. 2):

- Speech — **SHACER** (our Speech and HAndwriting reCognizER) — an ensemble speech recognition system that supports learning of out-of-vocabulary terms, whose component recognizers are based on highly modified versions of the Carnegie-Mellon University's (CMU's) Sphinx 2 speech recognizer [7];
- Ink — **NISSketch** — a commercial sketch recognizer, which also incorporates an internal interface to various handwriting recognizers;
- Pointing gestures — 3D Gesture — based on the recognizer used in our MAVEN system [6].

The core of our multimodal processing is an integration agent we call the multiparser. The multiparser is a temporal chart parser [8], receiving time-stamped messages from all connected input modes and integrating them to produce higher level constituents.

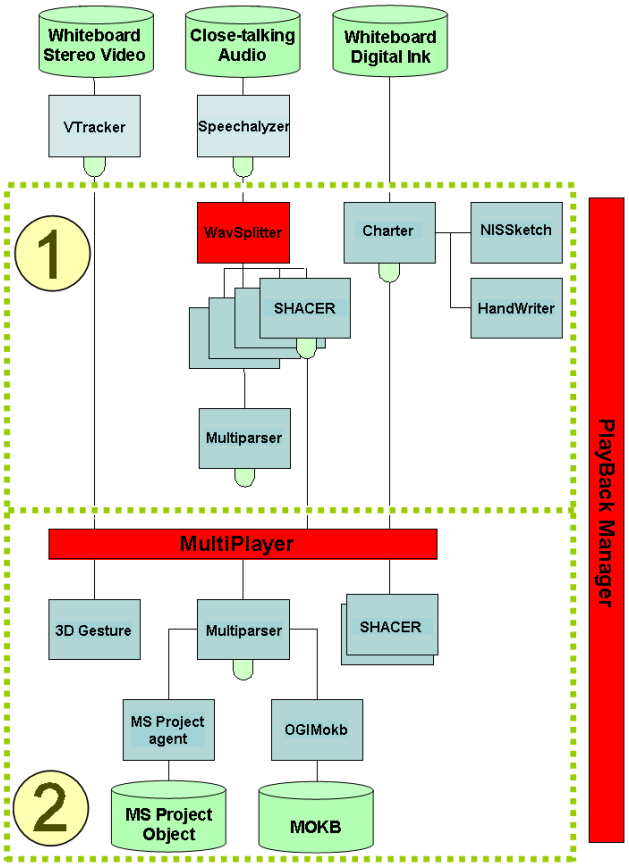

Figure 2: Components of the Charter system. Components in red are playback-specific. The green cylinders at the top and bottom edges represent intermediate files. The box labeled "1" corresponds to the first (unimodal) processing phase; the box labeled "2" indicates elements of the second (integration) phase.

2.3 Play Back Mechanisms

Play back processing in Charter is based on a two-phase process: 1) unimodal data replay and 2) integration replay. The first phase — unimodal data replay — runs individual recognizers against the raw speech, raw ink and raw tracked motion data streams. The results of these individual recognitions are stored in intermediate files following the general Task Discussion replay methodology (cf. Section 3). The second phase takes these intermediate results and plays them back through the core integration engine — multiparser (Figure 2).

Breaking processing into two separate phases, one concerned with unimodal processing and the second responsible for integration avoided the need of introducing potentially complex synchronization mechanisms into the recognizers themselves. Recognizers were mostly unchanged; except for changes to the way timing information was handled, as described below (Section 2.3.4). The need for complex synchronization would emerge from the differences in processing speed of the various recognizers involved, some of which are not able to process their input

streams in real-time. Their concurrent execution would then require the use of a protocol among all recognizers to guarantee that execution proceeded at the pace of the slowest recognizer. The split-phase approach we developed avoids this by allowing each recognizer to run at its own speed, isolating the synchronization into a single play back component run during the second phase, as detailed in Section 2.3.3.

2.3.1 Play Back of Audio Streams

Audio streams for each of the participants are processed by having a play back agent called WavSplitter (Fig. 2) extract speech-related segments from the raw audio files. To do this WavSplitter uses the endpoint information produced by CMU's Speechalyzer. It then retrieves the audio segment that was detected as speech and saves it in an intermediate memory cache. Each such segment is then passed to the ensemble of SHACER speech recognizers for processing, which results in a logged message from each recognizer holding timing information, recognizer ID and a phone transcription of the segment.

SHACER speech processing within Charter uses four low-level grammars: sequence constrained syllable and phone grammars, and unconstrained syllable and phone grammars. These grammars each run within a separate instance of the speech recognizer, thus an ensemble of four speech recognizers is used to perform an integrated phone level recognition pass. In parallel with the low-level phone recognition Charter also runs a special Word-Phrase Spotting Recognizer (WPSR) which is designed both to allow dynamic enrollment of newly recognized out-of-vocabulary (OOV) terms and also to use those dynamic additions to its grammar to spot instances of the enrolled terms in subsequent utterances, which contributes to improved system recognition at the multimodal level.

Processing using this ensemble takes longer than real-time. Furthermore, given that there usually are up to three audio streams to process, one for each of the participants of an interaction, one would need to execute concurrently three sets of four recognizers, adding up to twelve recognizers in all, if all streams were required to be processed in parallel. Despite the fact that the system is able to run these recognizers in a distributed fashion, this number of would require a number of machines and a number of audio soundcards that are not readily available in most sites.

Since each input stream is processed independently in the approach we employ for play back, processing can be performed piecewise according to hardware availability. Each audio stream is therefore processed in sequence during the first phase of (unimodal) processing, storing the results in files for subsequent integration in the second play back phase. Each play back phase is therefore composed of multiple processing steps (e.g. phase 1 play back is typically ink-to-sketch recognition processing followed by ensemble speech processing of each participant by each recognizer in turn). To facilitate the operation of the play back functionality, a process control mechanism was implemented, as reported in the following section.

2.3.2 Process Control

Play Back Manager (Figure 2) is the component that handles all the complexity involved in activating the different individual recognizers during each phase, and controlling the sequencing of execution. Play Back Manager embeds a state transition model that corresponds to the processing steps involved in each phase.

Play Back Manager locates the file location information that tells where each of the required streams are stored either by querying the global knowledge base (MOKB — Meeting Ontology Knowledge Base [9]), which, among other things, stores meta-information about locations of the raw streams related to a meeting. Alternatively, location information may be provided via a conventional dialogue-box user interface made available by Play Back Manager.

Components are then started via OAA (Open Agent Architecture [10]) messages that inform them of the location of their inputs, among other parameters. State transition within Play Back Manager is triggered by the reception of termination messages sent by each component after each respective play back message is processed. Besides starting processing as described above, the Play Back Manager embodies additional functionality to send out messages that set the state of the various recognizers in preparation for processing, such as loading up dictionaries or updating system internal database parameters for debug break point processing as described later in Section 2.4.

The current implementation represents the state transition model internally. A version is being designed that will make it possible to define this model in a configuration file to facilitate modification. It is also planned that this new version will be capable of launching and shutting down the modules automatically, without requiring the manual intervention that is currently needed.

2.3.3 Integration Play Back

The second play back phase — integration — is the one in which pre-processed unimodal information is run through the fusion mechanism, generating robust multimodal interpretations of the input data. In Charter, multimodal integration might occur for instance when speech recognition results are coupled with hypotheses generated from the handwriting recognition, and with aspects of the context to arrive at the spelling, pronunciation and semantics of the handwritten and spoken input [11]. In our system handwriting and speech occur together in this way when users are labeling constituents of the sketched diagram.

The main replay component involved in the integration phase is Multiplayer. Multiplayer's function is to extract individual elements from the files produced by the multiple recognizers run in the unimodal processing phase (phase one) and to send these elements out as OAA messages according to the order expressed in these elements' time stamps. Multiplayer has two modes of operation: real-time and lockstep. In real-time mode, Multiplayer introduces delays in between the messages that correspond to the relative time stamp offsets. In lockstep mode, Multiplayer waits for a signal that indicates that a message has been processed to completion before it sends the next one. Lockstep mode is essential for play back in situations, such as is the case in the current Charter system, in which the multimodal fusion itself takes more than real-time to process. The alignment of handwritten and spoken hypotheses and the corresponding second-pass speech recognition that takes place as part of our Multimodal Out-Of-Vocabulary Recognition (MOOVR) technique presently takes longer than real-time.

Successful use of lockstep mode requires that every component that receives a message from Multiplayer be able to signal completion of processing. If that fails to happen, execution blocks, as Multiplayer will only issue new messages once it has received this respective throttle control message.

2.3.4 Time-Related Adjustments

To equip individual recognizers to participate in the play back architecture the following adjustments were introduced with regard primarily to the way timing issues were handled:

- All input stream messages need to have a common initial time reference. Since it is in general not possible to start the recordings of the multiple streams at precisely the same time, it is necessary for each stream to be marked with the actual starting time so that they can be later aligned for replay. Some of the streams, such as the ink and the stereo vision images were individually stamped; the audio was stamped with a single initial time (embedded in the file name). Components were modified to use these original reference times instead of the machine clock time that would be used for real-time processing.
- Multiple machines are usually employed during recording. It is important for all machines to be synchronized as precisely as possible to guarantee that the individual time stamps collected are close enough to a single reference time. Network Time Protocol (NTP) was used for this purpose.
- Many of the higher-level recognition results in Charter are individually time stamped with beginning and ending times based on the elapsed processing time of their component lower-level recognized events (e.g. a semantic interpretation of speech has time-stamps based on how long it took to process the spoken utterance being interpreted). Multimodal fusion takes into consideration these higher-level begin/end time intervals in order to determine whether or not to integrate semantic elements of different modalities. In order to guarantee a uniform higher-level time reference equivalent to what would have been obtained if the system were run in real-time the initial higher-level event time is set to the low-level event start time from the moment of recording. The high-level ending time, however, is calculated not by the amount of processing time but rather is based on the amount of time covered by that segment of data being analyzed. For example the duration bounds of a semantic interpretation of speech would be those of the underlying audio segment. Since play back processing time varies (being faster or slower than real-time), the higher-level ending time cannot be assigned based on elapsed processing time.
- Some components, such as CharterUI and the Multiparser relied on real-time triggers for certain actions like the timed removal of stale information. These real-time triggers were replaced by a mechanism that bases its account of how time has elapsed on the time stamps of the latest received event, rather than on the actual machine clock time.

2.4 Debugging Support

As a chart parser multiparser can accept any context free grammar. For simplicity all rules for our system are in Chomsky normal form; that is, the left-hand side of each production is defined by exactly two right-hand terms. Some versions of our rules have a second right hand side member that is a list, and such rules are designed to carry along the remaining list members in each higher level constituent until the list is exhausted. A left-hand side is produced when both (1) its right-hand sides constituents unify with the left-hand side structure definition (*lhs-sd*) and (2) the temporal, spatial and other constraints defined in the *lhs-sd* are also satisfied.

Our multiparser rules are written in prolog and both unification of inputs and the length of time that integrated constituents stay on the chart is time-specific. Thus visualizing the flow of integration is difficult. Our prolog components use Quintus prolog, for legacy reasons, and its rudimentary visual debugging environment is both more verbose and more detailed than is practical for our debugging purposes. The primary needs of our debugging environment are checking rule-level unification and rule-level constraint satisfaction. The most common reason for rule non-integration is temporal constraint failures. In most cases such temporal constraint failures work to our advantage keeping the system tractable when temporal relationships are too distant. However, when rules have been specifically written to combine inputs that are known to occur temporally close it can be very time-consuming to collect appropriate recognition messages, combine them, and then debug their reasons for failure. This difficulty is compounded when there are tens or hundreds of seconds of system processing necessary to reach the contextual state in which a specific rule combination will fire. Some of the most elusive errors are those related to delayed recognition of inputs due to computational tractability issues. These are exactly the issues most likely to occur in the context of the non-real-time play back mechanism discussed in this paper. What we need from a multimodal debugger then is a way to run as directly as possible to a contextual state in which rule firing occurs and then a means of visualizing both the unification errors and spatial/temporal constraint errors that are causing failures. There are of course issues of debugging the logical flow of processing, but these too can be more transparently debugged when we can run quickly and unambiguously to the breakage points.

A further complication for debugging in our system is the fact that the various distributed agents may be written in any of several supported languages (e.g. C, C++, Java, Prolog, Python, etc). Thus when setting breakpoints during play back we need to be able to step off into not only prolog agent code but also C, C++ or Java agent code as well. Thus our debugging environment cannot be handled within a single Integrated Development Environment (IDE). We must use several IDEs each appropriate for its respective agent language and then communicate between these IDEs with individual and scripted OAA control messages. Our goal is then, as is the goal for any debugging environment, to display the right amount of information at the right time so we can more quickly come to understand the reasons for system failures.

The critical elements of our debugging approach are two-fold: (1) real-time simulated playback in lockstep mode, and (2) a way to reset the system at a reasonable and useful level of granularity to avoid having to play back long sequences of inputs to reach a failure state that is being debugged. Using the mechanisms we have put in place we have been able to reduce our debugging time several fold and in some instances even by an order of magnitude (e.g., debugging tasks that took days or even weeks of

collaborative thought and effort can now be debugged in a matter of hours).

To accomplish (1) above requires an adequate data collection superstructure (cf Section 3) capable of producing the synchronously time-stamped multimodal log files, which are the basis of real-time simulated playback. To accomplish (2) requires identifying all the elements of system state that exist during processing, designing ways to capture and log that state at appropriate levels of granularity, designing and implementing a meta-controller to steer the loading of state and subsequent real-time simulated playback, callback mechanisms to control the lock-step real-time simulation (so that non-real-time processes are not overwhelmed by a flood of inputs), a means of setting break points, and then locally callable mechanisms for turning various aspects of debugging visualization on or off at the specified break points.

Our eventual goal is a visual multimodal debugger that lets one see the inputs as they arrive and watch their unification and constraint interactions as they are processed. At present there is no single IDE capable of this level of visualization across our heterogeneous agent code base. So instead we use our Multiplayer (Section 2.3.3 above) running in debug mode as a base IDE, in which we can watch the arrival of input messages and place break points that fire at the arrival of specific messages. The input to our Multiplayer IDE is the log file resulting from phase 2 processing (Section 2.3.3 above). Since this phase 2 output result log file is the basis for debugging play back it contains not only all semantic level interpretations and integrations that occurred during phase 2 play back, but also snap-shot information of the system state at every change-point that occurs during playback. Such change-points occur whenever a recognizable input within the meeting recognition space occurs. Recognizable inputs are those that result in a change on the charter UI screen, causing the sketch artifact to be redrawn.

As well as phase 2 integration messages, some messages from phase 1 play back are passed directly through to this phase 2 log. For example, phonetic transcript interpretations from only three of the four ensemble phone recognizers are logged. These are the ensembles' slave recognizers, while the forth ensemble recognizer is the master. The speech processing message that sends audio input to the master must be logged and then replayed during debug play back, because the master must take the audio segment as input in order to process it, cache its MEL-cepstrum features, and then use them later in a second-pass recognition in response to a request from the multiparser to combine (a) letter-to-sound transformations of the handwriting alternatives with (b) the phone transcripts from the three slave recognizers, as well as transcript alternatives from (c) the Word-Phrase Spotting Recognizer (WPSR) and from (d) the large vocabulary continuous speech recognizer running in parallel within CMU's Speechalyzer component.

If successful in combining the inputs from the multiparser request the master produces a semantic level interpretation of the combined handwriting/speech event, yielding the spelling, pronunciation and semantics of a chart constituent label, which may either be a new term, a new pronunciation, or a disambiguation of an existing term. In integrating this combination the system employs a kind of short-term memory of the spoken inputs (in sliding window caches of MEL-cepstrum features, WPSR transcripts, and Speechalyzer transcripts) because it is possible that a handwritten term would best align not with the closest speech segment but rather with an earlier one. Practically speaking this calls for setting breakpoints at the system snapshot one utterance before the one that is to be debugged, to allow the sliding window buffers to be properly filled when the breakage point is reached.

To see how we can set a play back breakpoint for debugging let's look at an example of debugging the combination of a handwriting/speech event that occurs five minutes into a recorded meeting. Since every system state was logged as a block of snap-shot information messages within the phase 2 result log we first remove the portion of the log file preceding the system-state snap-shot at which we want to begin, then we remove all system-snapshot information succeeding that — keeping only the messages to be played back for integration (this processing is automated at the script level). That way during the debug play back run the system will first reset all relevant state (skipping the first five minutes of play back and directly setting the system to the resulting state of that integration), then resume integration of succeeding messages just as they would have been played back during phase 2. Our break point mechanism lets us specify the text of any message in the processed log file and stop execution in the Multiplayer IDE when that message is about to be processed. In this case we'll play back at least the utterance preceding the one we want to debug (to prime the sliding window cache), then play back the handwriting interpretation, and finally break at the spoken utterance we've flagged as a break point. At this point we can use our MessageTool to send an OAA message or messages that set the debug state within the prolog code of our multiparser agent suite. We can set a debug state that shows unification failures, a debug state that displays spatial/temporal constraint processing, or both. It is usually helpful to display the least amount of information necessary. We then play back the utterance message and examine the resulting output to determine where the failure has occurred. If the problem is not in the prolog agent code, then we can set a break point in the IDE running the debug version of the master SHACER speech recognizer, send it paused message from Multiplayer, and then step through the speech agent code to discover where the combination failure occurs.

In summary by keeping state snap-shots during phase 2 play back we can support fast replay of a meeting by jumping immediately to any given system-state snap-shot and proceeding with message play back from that point forward. Using our Multiplayer IDE we can set break points on any given log message and then branch to appropriate debugging choices: either message-switched displays of low-level unification and/or constraint processing information in prolog code, or via break points in other agent IDEs (like C, C+ or Java debuggers) be able step through agent code to which the Multiplayer messages cause input to be directed.

3. THE TASK DISCUSSION SYSTEM

The Task Discussion system in which the Charter Suite is embedded provides a set of additional services related for instance to topic segmentation [12], role detection [13], activity detection and face recognition [14] and summarization.

The main objective for the play back of this system is to collect data for the yearly evaluation of the system. In this section we describe the collection environment (Section 3.1), overview the

system architecture and replay process (Section 3.2) and describe the testing procedure (Section 3.3).

3.1 Collection Environment

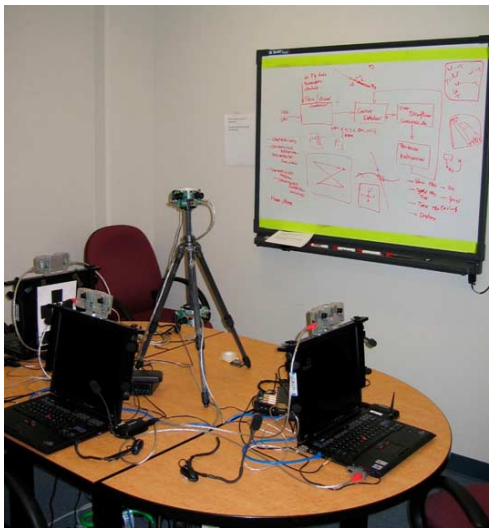

Figure 3: View of the CALO Task Discussion data collection meeting room showing interactive whiteboard, the laptop-frame-mounted stereo cameras, and CAMEO.

The CALO Task Discussion system is constructed to precisely record data from various sensors (audio, video, digital ink, and instrumented applications) available in a data collection room (Figure 3). The keys to the data collection are to accurately record all data streams with precise timestamps, to ensure that all important data is captured, and to limit biases in the data capture based on the needs of the current recognizers. The collection environment is made up of the following components.

- *Frame* (stereo-vision camera and 4-microphone array attached to laptop), 1 per meeting participant. The frame is used for face tracking and face orientation recognition.
- *Close-talking microphone* (along with text notes and MS PowerPoint instrumentation), 1 per meeting participant. Used to collect speech for transcription and summarization, and to provide spoken input for multimodal project schedule diagram creation.
- *CAMEO* (omni-directional video situated in the center of the room), 2 per meeting positioned at various heights. A 360° view of the room is provided by CAMEO [14]. This is used to track participants as they move around the room, recognizing faces and basic activities (e.g. standing, sitting).
- *Camcorder* (in corner of the room with attached omni-direction microphone situated in the center of the room).
- *Instrumented whiteboard*. The instrumented board is used for input of sketched project schedule diagrams interpreted by the system.
- *Stereo camera* over the instrumented whiteboard (not shown in Figure 3). This camera tracks gestures made towards elements sketched on the board.
- *Gigabit Ethernet* connecting all computers in instrumented meeting room
- *Storage Computer* with 1/4 terabit hard-drive accessible to all computers for archiving raw data and recognized results.

Participants' laptops, on which frames are mounted and to which the individual microphones are connected also run instrumented versions of applications, so that the system has access for example to notes that are input during a meeting, or to slides that are being presented. Network Time Protocol (NTP) is used to keep the clocks of all computers synchronized.

The goal was to record both the audio and video at high frame rates with high resolution then down-sample as desired. Unfortunately, this was not performed. Instead the audio was captured at 11 KHz (according to the current automatic speech recognition needs) and the video was recorded as fast as the machines permitted (usually 15 frames per second).

3.2 Architecture and Play Back Process

The system uses the Open Agent Architecture (OAA) framework [10] as the underlying communication mechanism. The agent-based recognizers register what data stream(s) they consume and how that data is consumed (simulated real-time or offline). Their input needs are used to construct a directed acyclic graph (DAG), which is currently created manually, but will in the future be created automatically with proper registration of data inputs and data outputs.

The DAG is traversed by selecting a recognizer to run using an operator GUI currently implemented with a servlet, which allows it to be run from any browser enabled computer. This interface lists instructions and provides a link that generates the required OAA message to get a phase started. Currently, it is up to the operator to guarantee that all the necessary inputs are available, and to ensure that all agents that will handle the request are running.

Not only can a single meeting be processed in this offline DAG traversal format but a series of meetings can be processed as well. In this case each agent has both its standard input channels as well as any past meeting results. The past results include not only the results from when it processed the meeting but also include any changes made to those results either by later downstream agents or by the user. With this added feedback, the recognizers have the capacity to learn over time across the series of meetings.

3.3 Result Storage and Testing

The persistent semantic results of the execution of the system are stored by each component in a global knowledge base - the Meeting Ontology Knowledge Base (MOKB) [12]. Knowledge inserted into MOKB complies to a multimodal ontology and can be queried during or after processing. Various tools can either present the results to the operator for visual inspection or can automatically compare the results with past results. Our interface queries MOKB and displays transcripts, project schedule recognition results, notes collected by the instrumented applications and other information resulting from processing. In the current system, the same servlet that is used to control the replay can also display cumulative summary results in html format.

To test the system, a variety of questions were created that a typical executive's assistant would be able to answer with regard to the meetings. These questions were designed to take various parameters (so they are referred to as parameterized questions or

"PQs") effectively expanding the finite question template set to a nearly infinite set. The research team was told the set of PQs, to help direct the research efforts, but was not told how those questions would be instantiated (where instantiated questions became known as "IQs").

To administer the test, users were told to participate in a set of five meetings. After the meetings took place, a random sampling of PQs was selected and each question was instantiated, sometimes multiple times. The meetings were then annotated with these IQs in mind to determine the gold standard answer to each question. To determine actual system learning, the test was given twice: first testing the system on the fifth meeting after it had processed all five meetings (giving it the chance to learn context over the course of the meetings) and then resetting the system and testing it again on the fifth meeting but without allowing it to first process the initial four meetings. The goal of testing was to show the system's ability to perform better due to the effects of learning from the earlier meetings.

To actually answer each IQ, the test agent sent each ontological query to the Query Manager (QM). The QM, with its centralized knowledge of where all CALO knowledge resides, broke each question down into their constituent parts sending requests to the various data stores (sometimes the user's personal CALO knowledge base and sometimes the meeting knowledge base) and then performed reasoning over the results prior to sending the answer back to the test agent. Agents that received the QM's requests also could perform reasoning over them if necessary. Thus the test was not simply a static lookup into a database but both during the meeting processing phase and question answering phase a true cognitive system.

To support answering "what if" questions, the test agent directly accessed the knowledge base and inserted new facts (checked by hand to make sure they were valid) as part of a per question setup phase. The question was then sent to the Query Manager for answering and a cleanup stage reset the knowledge bases to their state prior to asking the question so as not to affect future questions. Using this approach to testing, an unbiased test was given that accessed the Task Discussion system's capabilities and determined which of those capabilities were due to learning. Thus our test facility focused on measuring learning in contrast, for instance, to the recent Evaluation Packages for the First CHIL Evaluation Campaign which focuses on scores for individual recognition tasks [15].

4. COLLECTION AND PLAY BACK FOR ANNOTATION AT OGI

The data collection environment at the Oregon Graduate Institute captures and synchronizes five high-resolution videos with four audio sources. The videos are captured by Pt. Grey Scorpion firewire cameras. Resolution ranges from 1024x768 to 1280x960 at 15 to 20 frames per second. Three of the microphones are Countryman ISOMAX close-speaking microphones. The fourth microphone is a room microphone.

The video streams are captured on three Unix (Fedora Core) machines. The machine clocks are synchronized via the Network Time Protocol (NTP). Each video buffer is compressed by quad-tree compression, which results in about 50% (lossy) compression. Then each frame is appended with a header containing, among other things, an absolute timestamp. Each of these frames is written to hard disk in what we call a DAT file. Every minute, a new DAT file is created, as the file size would quickly exceed the 4 gigabyte maximum otherwise.

Audio buffers are captured via Mobile-Pre MAudio external analog-to-digital converters (ADC), which plug into a USB port. Using an external ADC as opposed to an internal soundcard eliminated electrical interference noise introduced by the machine. Audio buffers are also appended with a header containing an absolute timestamp. It is vital that there are no dropped audio buffers as that would degrade the synchronization. In earlier studies, where audio quality wasn't critical, we identified dropped audio buffers by interpolation of timestamps and filled with empty buffers. However, audio quality became very important in later studies, and as a result we moved audio capture from the video capture machines onto two Windows XP machines, also synchronized via NTP. DirectSound automatically buffered the sound coming from the ADC's, resulting in no dropped audio frames.

By recording data in this format, with a header containing rich meta-data for every buffer, we can synchronize video and audio with any input source which has a clock synchronized via NTP. Examples include touch-sensitive white boards, digital paper, screenshots from a computer, etc.

These intermediate DAT files are converted to universally viewable MPEG-4 media streams off-line. A video stream is paired with an audio stream to form a single media stream. There is synchrony between the audio and video of a single media stream and across multiple media streams in a single session. This is achieved by stepping through all the DAT files (video, audio, digital ink, etc.) of a particular session and identifying a unified start position. A wave file is created from the audio DAT file. The video buffers are converted to YUV frames and piped along with the wave file to an open source video encoder called ffmpeg. The result of this process is X synchronized media streams.

For playback and annotation we modified an existing annotation tool, named MockBrow [16], developed at Carnegie Mellon University. This tool, written in Java, utilized the Java Media Framework (JMF) for playing media. The JMF is not intended to play multiple high-resolution videos and couldn't handle the load. We needed a media framework that was not abstracted so far from the hardware. We extended the JMF with a wrapper around a DirectX videoplayer we developed. Function calls were funneled through the Java Native Interface (JNI). The switching of video players was transparent to MockBrow. In this way, we could instrument any annotation tool utilizing the JMF with our optimized video player.

5. RELATED WORK

Multimodal integrated development environments (MMIDEs) are not yet common; however, based on standards work at the W3C® IBM now markets a Multimodal Toolkit and Browser, which uses XHTML+Voice (X+V for short – a multimodal markup language) to support high-level, rapid prototyping and creation of multimodal (speech + graphics) Web applications [17]. Such a Toolkit is not designed to support debugging at the recognition module level, such as we have described in this paper. Its simplicity and ease of use derive from its homogeneous description language, X+V, while research multimodal systems tend to be large, collaborative efforts that are very heterogeneous

in terms of their component code modules. For example, Multiplatform [18], an integration testbed used by the Smartkom system employs its own multimodal markup language called M3L as a communication protocol between the 40 or so component modules (written for 3+ platforms in 5+ languages) of which it is made up. OAA (the Open Agent Architecture) [10] which is the basis of communication in our system also uses a multicast publish/subscribe architecture (e.g. blackboard) to facilitate the use of heterogeneous modules. Our work uses that architecture as a starting pointing and offers an implementation approach, based on OAA message tracking, to support the eventual evolution of a genuine MMIDE for heterogeneous modules. As the ICARE project has done in using Petri nets to formally model multimodal systems in ways that allow explicit testing of usability and reliability [19] we will need to better explore formal engineering techniques and processes as we move forward.

6. CONCLUSIONS AND FUTURE WORK

The play back mechanism of a complex multimodal system – the Charter Suite – was presented. Replay in Charter is handled in two phases: 1) individual unimodal recognition of each data stream and 2) multimodal integration. The second phase of replay – integration – introduces mechanisms for lockstep execution via the Multiplayer component. This mode of execution allows for non real-time integration by delaying the generation of messages based on a throttling signal. We argued that this design lowers the synchronization complexity of the system, decreases the amount of modifications required for system recognizers, and allows for restarting the play back at any convenient point. The complexities introduced by the need to control the execution of multiple individual steps within each phase have been addressed by our PlayBackManager component, which initiates component execution based on a state transition model. Ongoing work is required to make that process control more flexible by allowing for instance the use of external configuration files to determine the state transition model that is employed. We have illustrated two different uses of replay – one for the purpose of system evaluation, and the other for manual annotation of multimodal data.

7. ACKNOWLEDGMENTS

The authors would like to warmly thank Lynn Voss of SRI for his generous contributions to this paper. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA or the Department of Interior National Business Center (DOINBC).

8. REFERENCES

[1] CALO, *http://www.ai.sri.com/project/CALO*.

[2] Kaiser, E., et al. *Demo: A Multimodal Learning Interface for Sketch, Speak and Point Creation of a Schedule Chart*. in *International Conference on Multimodal Interfaces (ICMI '04)*. 2004. State College, PA.

[3] Kaiser, E.C. *Multimodal New Vocabulary Recognition through Speech and Handwriting in a Whiteboard Scheduling Application*. in *Proceedings of the International Conference on Intelligent User Interfaces*. 2005. San Diego, CA.

[4] Barthelmess, P., et al. *Distributed Pointing for Multimodal Collaboration Over Sketched Diagrams*. in *The Seventh International Conference on Multimodal Interfaces (ICMI '05)*. 2005. Trento, Italy.

[5] Cohen, P.R., et al. *QuickSet: Multimodal Interaction for Distributed Applications*. in *International Multimedia Conference*. 1997.

[6] Kaiser, E., et al. *Mutual Disambiguation of 3D Multimodal Interaction in Augmented and Virtual Reality*. in *International Conference on Mutimodal Interfaces (ICMI '03)*. 2003.

[7] Singh, R., *The Sphinx Speech Recognition Systems*, in *Encyclopaedia of Human Computer Interaction*, W. Bainbridge, Editor. 2004, Berkshire Publishing Group.

[8] Johnston, M., et al. *Unification-based Multimodal Integration*. in *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*. 1997: Association for Computational Linguistics Press.

[9] Niekrasz, J., et al. *Ontology-based Discourse Understanding for a Persistent Meeting Assistant*. in *In Proceedings of the 2005 AAAI Spring Symposium on Persistent Assistants*. 2005.

[10] Martin, D., A. Cheyer, and D. Moran, *The Open Agent Architecture: A Framework for Building Distributed Software Systems.* Applied Artificial Intelligence, 1999. **13**(1/2).

[11] Kaiser, E.C. *Dynamic New Vocabulary Enrollment through Handwriting and Speech in a Multimodal Scheduling Application*. in *Making Pen-Based Interaction Intelligent and Natural, Papers from the 2004 AAAI Symposium, Technical Report FS-04-06*. 2004. Arlington, VA., USA.

[12] Niekrasz, J. and M. Purver. *A Multimodal Discourse Ontology for Meeting Understanding*. in *The 2nd Joint Workshop on Multimodal Interaction and Related Machine Learning Algorithms*. 2005. Edinburgh, Scotland.

[13] Banerjee, S. and A. Rudnicky. *Using Simple Speech-Based Features to Detect the State of a Meeting and the Roles of the Meeting Participants*. in *In the Proceedings of the 8th International Conference on Spoken Language Processing (Interspeech 2004 - ICSLP)*. 2004. Jeju Island, Korea.

[14] Rybski, P.E., et al. *CAMEO: Camera Assisted Meeting Event Observer*. in *In Proceedings of ICRA-2004*. 2004. New Orleans.

[15] CHIL, *Evaluation Packages for the First CHIL Evaluation Campaign (http://chil.server.de/servlet/is/2712/)*.

[16] Banerjee, S., C. Rose, and A.I. Rudnicky. *The Necessity of a Meeting Recording and Playback System, and the Benefit of Topic--Level Annotations to Meeting Browsing*. in *In the Proceedings of the Tenth International Conference on Human-Computer Interaction*. 2005. Rome, Italy.

[17] IBM, *Developing applications using the Multimodal Tools (ftp://ftp.software.ibm.com/software/pervasive/info/multimodal/developing_applications.pdf), White Paper,* 2003.

[18] Herzog, G., et al. *MULTIPLATFORM Testbed: An Integration Platform for Multimodal Dialog Systems*. in *HLT-NAACL 2003 Workshop: Software Engineering and Architecture of Language Technology Systems (SEALTS)*. 2003. Edmonton, Canada.

[19] Bastide, R., et al. *A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts*. in *International Conference on Mutimodal Interfaces (ICMI '04)*. 2004. State College, Pennsylvania.