GSI DEMO: Multiuser Gesture / Speech Interaction over Digital Tables by Wrapping Single User Applications

Edward Tse[1,2], Saul Greenberg[1], Chia Shen[2]
[1]University of Calgary, [2]Mitsubishi Electric Research Laboratories
[1]2500 University Dr. N.W, Calgary, Alberta, Canada, T2N 1N4
[2]201 Broadway, Cambridge, Massachusetts, USA, 02139
[1](403) 210-9502, [2](617) 621-7500
[tsee, saul]@cpsc.ucalgary.ca, shen@merl.com

ABSTRACT
Most commercial software applications are designed for a single user using a keyboard/mouse over an upright monitor. Our interest is exploiting these systems so they work over a digital table. Mirroring what people do when working over traditional tables, we want multiple people to interact with the tabletop application and with each other via rich speech and hand gestures. In previous papers, we illustrated multi-user gesture and speech interaction on a digital table for geospatial applications – Google Earth, Warcraft III and The Sims. In this paper, we describe our underlying architecture: GSI DEMO. First, GSI DEMO creates a run-time wrapper around existing single user applications: it accepts and translates speech and gestures from multiple people into a single stream of keyboard and mouse inputs recognized by the application. Second, it lets people use multimodal demonstration – instead of programming – to quickly map their own speech and gestures to these keyboard/mouse inputs. For example, continuous gestures are trained by saying "Computer, when I do [one finger gesture], you do [mouse drag]". Similarly, discrete speech commands can be trained by saying "Computer, when I say [layer bars], you do [keyboard and mouse macro]". The end result is that end users can rapidly transform single user commercial applications into a multi-user, multimodal digital tabletop system.

Categories and Subject Descriptors
H5.2 [**Information interfaces and presentation**]: User Interfaces. – Interaction Styles.

General Terms: Algorithms, Human Factors.

Keywords: Digital tables, multimodal input, programming by demonstration.

1. INTRODUCTION
A burgeoning research area in human computer interaction is digital table design, where natural table interaction is recreated and extended in ways that let multiple people work together fluidly over digital information.

We already know that the traditional desktop computer is unsatisfying for highly collaborative situations involving multiple co-located people exploring and problem-solving over rich spatial information, e.g., [3]. These systems inhibit people's natural speech and gesture interaction over paper-based tabletop information such as maps (e.g., Figure 1) [3][9]. A critical factor is that most digital systems are designed within single-user constraints. Only one person can easily see and interact at any given time. While another person can work with it through turn-taking, the system is blind to this fact. Even if a large high resolution display is available, one person's standard window/icon/mouse interaction – optimized for small screens and individual performance – becomes awkward and hard to see and comprehend by others involved in the collaboration [8].

Figure 1. People interacting over a paper map on a tabletop

In our previous work, we illustrated how we could enable interaction with single user applications through speech and gesture over a multi-user digital table, e.g., as illustrated in Figures 2 and 3. In particular, we offered a set of behavioural foundations motivating collaborative multimodal interaction over digital tables [11]. We then presented several case studies, listed below, that illustrate how we could leverage the power of commercial single user applications to the digital table.

- *Maps.* We used Google Earth (earth.google.com) as an exemplar of how people could collaboratively search, navigate, bookmark, and annotate satellite imagery (Figure 2). Collaborators could use gestures to continuously zoom and pan the digital map to adjust the focal area. Using speech,

Cite as:
Tse, E., Greenberg, S. and Shen, C. (2006) GSI DEMO: Multiuser Gesture / Speech Interaction over Digital Tables by Wrapping Single User Applications. Report 2006-833-26, Dept. Computer Science, University of Calgary, Calgary, Alberta, CANADA. May. Also as MERL Technical Report TR2006-46, Mitsubishi Electric Research Laboratories, Cambridge, MA, 02139, USA.

Figure 2. Two people interacting over a digital map on a table

they could jump to discrete locations by saying "fly to [location]", and could rapidly overlay geospatial information by the "layer [information]" speech command [11].

- *Command and Control.* We transformed Blizzard's Warcraft real time strategy game (www.blizzard.com/war3) into a form that illustrated how collaborators could use gesture and speech to control actions of troops over a geospatial landscape, e.g., "move here [point]" (Figure 3) [11][12].
- *Simulation.* We repurposed Electronic Arts' The Sims (thesims.ea.com) to let collaborators use gesture and speech to navigate a virtual home, to create, reposition and manipulate objects within that home, and to enable simulation in how the virtual people live within it [12].
- *Games.* While we used Warcraft and the Sims as 'toy' examples of purposeful applications, they also illustrate how single user games can be easily repurposed for multiplayer multimodal play over a tabletop display [12].

In this paper, we focus on the underlying architecture of our system, which we call GSI DEMO - *Gesture and Speech Infrastructure created by Demonstration*. GSI DEMO offers (1) multimodal programming by demonstration, and (2) a multi-user speech and gesture input wrapper around existing mouse/keyboard applications. A key research contribution of GSI DEMO is to enable people to use multimodal by demonstration – instead of programming – to quickly map their own speech and gestures to these keyboard/mouse inputs. One trains the system by demonstrating the gestures and speech actions it should recognize, and then performing the appropriate keyboard and mouse events it should play back. The result is that end users can quickly and easily transform single user commercial applications into a multimodal digital tabletop system. This multimodal programming by demonstration relies on a run-time wrapper around existing single user applications. This wrapper accepts speech and gestures from multiple people working over the table, compares them to a list of allowable actions, accepts the ones that have a reasonable match, and then translates these into a single stream of keyboard and mouse inputs recognized by the application. While some parts of the architecture of this wrapper were described in [11], what is presented here is much more detailed, and is more generalizable as it can immediately repurpose a broad variety of applications.

Figure 3. People using gesture and speech to play Warcraft III

We begin with a high level overview of the GSI DEMO architecture. This is followed by a description of our gesture / speech recognizer and unifier. We then show how people map gesture and speech to application actions by demonstration, and how these are invoked after recognition. We close with a brief discussion of the strengths and limitations of our approach.

2. GSI DEMO OVERVIEW

As mentioned, GSI DEMO is designed to allow end users to rapidly create, instrument and use their own multimodal gesture/speech input wrappers over existing single user applications [11][12]. However, it is not a generic gesture engine or speech recognizer. Its main capabilites are summarized below.

Gesture recognition. GSI DEMO focuses on gestures that model the basic everyday acts seen in tabletop use, e.g., [3][9][11]. Examples include one-finger pointing for selection, dragging and panning, two-handed multiple object selection by surrounding an area with upright hands [13], two finger stretching to zoom in and out of a region, fist stamping to create new objects, palm-down wiping to delete objects. These gestures have to be easily understood by others, as they also serve as communicative acts [11]. Consequently, GSI DEMO does not readily support complex and arbitrary gestures that act as abstract command surrogates, e.g., a D-shaped gesture over an object to indicate 'Delete'.

Speech recognition. GSI DEMO emphasizes recognizing simple discrete speech utterances that match a fixed vocabulary of commands (e.g., 'fly to Boston', 'stop'…). It monitors continuous speech is monitored for occurrence of these utterances.

Speech in tandem with gestures: GSI DEMO recognizes combinations of speech and gestures, where gestures can qualify a person's speech acts. For example, a speech command 'Create a piano' can be followed by a [point] gesture that indicates where that piano should be created.

Floor control. GSI DEMO also recognizes that near-simultaneous gestures and speech performed by multiple people may be combined, interleaved and / or blocked. It does this by supplying a set of appropriate floor control mechanisms that mediate how the system should interpret turn-taking.

Input surfaces. GSI DEMO is constructed in a way that handles multiple types of touch surfaces. Our current version handles the DiamondTouch Surface and the DViT Smart Board. Each device

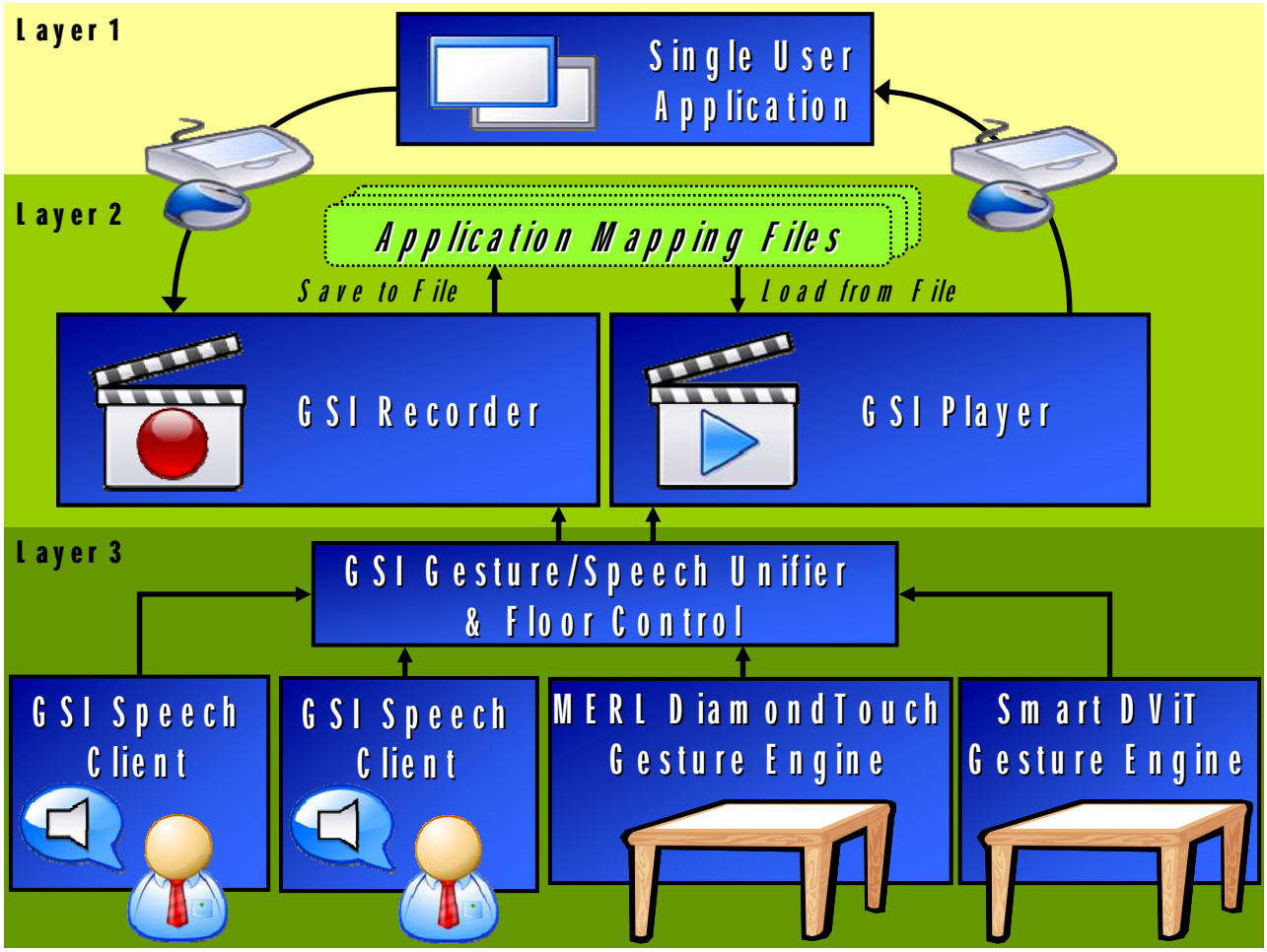

Figure 4. The GSI DEMO Infrastructure

offers different input capabilities, thus allowing different (and not necessarily compatible) types of gestures. They also differ in how inputs from multiple people are disambiguated.

Multimodal training. People teach the system by demonstration, where they map their own speech and gestures to keyboard/mouse inputs recognized by the single user application.

As seen in Figure 4, GSI DEMO is roughly composed of three layers. The first layer contains all of the low level gesture and speech recognition systems. Each GSI Speech Client, one per person, recognizes speech commands spoken by each individual working over the table. The MERL DiamondTouch and the Smart DViT Gesture Engines recognize hand gestures done atop two different types of touch surfaces. The speech and gesture commands of multiple people are combined (with appropriate floor control mechanisms to mitigate problems arising from overlapping actions) in the GSI Gesture/Speech Unifier and converted into unique commands that will be used to activate the keyboard and mouse mappings. The second layer defines the actual mapping of speech/gesture commands to actions understood by the single user application. Through the GSI Recorder, end users demonstrate multimodal speech and gestural acts, and then demonstrate the corresponding mouse and keyboard commands that should then be performed over that application. The Recorder generalizes these actions, and maps them together as a command. It then saves all that information to an application mapping file, where files can be created for different applications. For a given application, the GSI Player loads these mappings from the appropriate file. It then performs the corresponding keyboard and mouse commands when a speech or gesture command has been recognized by clients in Layer 1. Finally, the third layer is the single user commercial application that is oblivious to the recording and playback of multimodal speech and gesture actions of multiple people on the table top. From its perspective, it just receives mouse and keyboard input events as if they were entered by a single person.

3. SPEECH / GESTURE ENGINE

This section describes how the various components in Layer 1 work and interact.

3.1 GSI Speech Client

The GSI Speech client is responsible for speech recognition. As described below, it separates conversational speech from computer commands, disambiguates speech from multiple people, and has a few simple strategies for dealing with noise. Its output

is a command, which is the best match of the spoken utterance against a list of possible computer commands.

Input hardware. Our speech recognition hardware uses a combination of high quality wired lavaliere microphones along with several wireless Bluetooth microphones. Microphones are noise-cancelling, so that only the speech of the person closest to the microphone is heard. Wireless microphones are definitely preferred, as this lets people walk around the table.

Speech recognition software. We use an existing off-the-shelf speech recognition system: the Microsoft Speech Application Programmers' Interface (Microsoft SAPI). However, SAPI and other similar off-the-shelf speech recognizers are designed to support one person using a computer in a relatively quiet environment. Co-located tabletop environments differs in 3 ways.

1. *The aural environment is noisy*: multiple people may speak at the same time, and the constant actions of people around the table introduce additional sounds.
2. *Not all utterances are directed at the computer*: most of the speech is actually normal conversation between participants.
3. *Multiple people may be trying to direct the computer*: we need to recognize and disambiguate speech from multiple people.

In light of these circumstances, we designed a specialized application around the Microsoft SAPI system called the GSI Speech Client, whose graphical interface is shown in Figure 5. Its job is to manage speech recognition gathered from multiple people working on the table display, where it delivers possible commands that these people are trying to invoke over the single user application.

First, we simplify recognition issues in the presence of noise by giving each person a separate microphone headset; people select the microphone they are using from the Input Device list (Figure 5, bottom left). When a person speaks, the client matches that person's speech utterances against a dynamically reconfigurable list – a menu – of speech commands that should be understood (as described in the Application Mapping File, see §GSI Recorder). This list is visible in Figure 5, top half. If the match is good, it offers that menu command as a working hypothesis of what the person has said. In contrast to free speech recognition, this simple matching of utterance to a list element is considerably more reliable in the presence of noise, and suits our purpose of matching speech to simple commands that can then be invoked.

Second, we disambiguate natural conversations from speech commands directed to the computer in three ways. Our first strategy uses microphones containing a switch; people turn it on when commanding the computer, and off when talking to each other. This is problematic because people forget, and because switching actions interfere with people's ability to gesture atop the table. Our second strategy is based on the informal observation that people speak louder and clearer when issuing speech commands to a computer as compared to when they are speaking to each other across the table. We leverage this by providing a user-modifiable *Minimum Volume Threshold* (Figure 5 middle, current volume level is shown atop of the adjustable trackbar). Any talk below this threshold is considered interpersonal conversation, while talk above this threshold activates an attempt to recognize the current utterance as a command. Our third strategy, inspired from Star Trek TNG, has actors direct commands to the computer by prefixing it with the key speech phrase 'Computer'. In GSI Speech, this phrase is

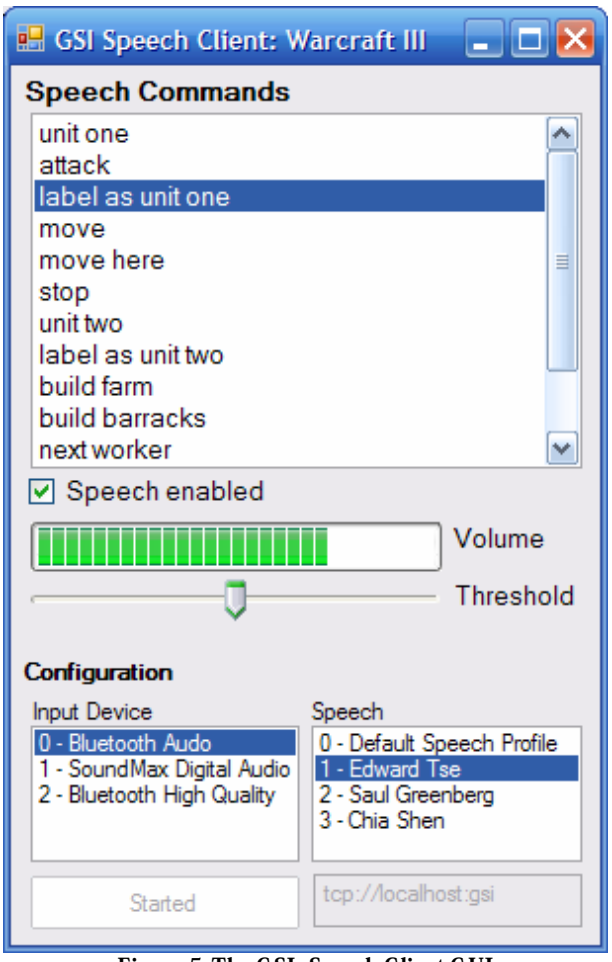

Figure 5. The GSI: Speech Client GUI

configurable, although we too use the 'Computer' prefix. Thus 'Computer: Label as Unit One' is considered a speech command, while 'Label as Unit One' is conversation. Regardless of the strategy, the computer indicates recognition by highlighting the item on the list in Figure 5, and by playing a short sound; this audio feedback suffices to let people know that the computer has attempted to interpret their speech. In practice, the number of false speech command recognition errors is significantly reduced when a speech command prefix is used.

Third, we manage multiple people by supplying multiple independent speech recognizers. Most 'out of the box' speech recognizers operate as single-user systems, i.e., one cannot easily plug in multiple microphones and expect the speech recognizer to disambiguate between them. We have two strategies for managing this. One option is to use multiple computers, one per person, where each operates its own independent SAPI speech recognizer. The GSI Speech Client also runs on each computer, and collects all audio volume, hypothesis and speech recognition information. It posts this information to a distributed data structure seen by the main GSI DEMO client. The second option creates multiple speech recognizers on a single machine, where particular microphones are directed to particular speech recognizers. As before, information collected by each process is posted to a distributed data structure. The tradeoff between these options is CPU load

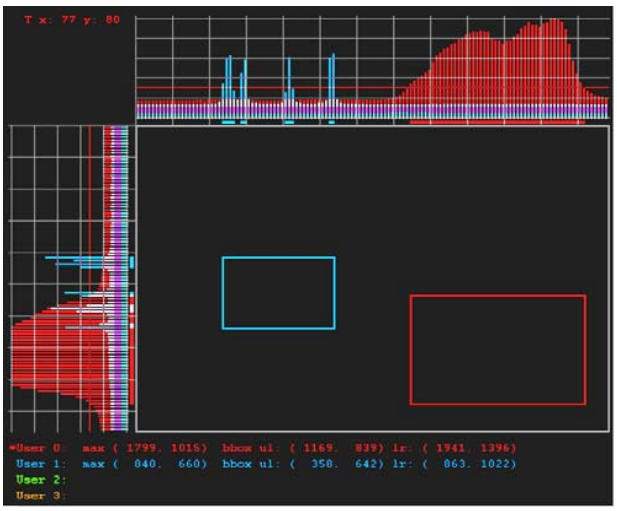

Figure 6. DiamondTouch signal, 2 people

(when multiple recognizers run on a single machine) vs. the complexity and cost of having multiple computers. The GSI Speech Client is flexible enough so that one can mix and match. Regardless of the method chosen, the client lets a person select their custom speech profile previously created using Microsoft SAPI (Figure 5, bottom right).

As mentioned, each speech recognition results in information posted to a distributed data structure. We use the GroupLab Collabrary to manage this sharing of information between processes and across multiple machines [1]. Each client posts the following information for every utterance it recognizes:

```
/{ClientId}/Menu = "unit one, label as unit one, ..."
/{ClientId}/SpeechEnabled = true
/{ClientId}/AudioLevel = 0 (min) to 100 (max)
/{ClientId}/Recognition = 1
```

The `ClientID` is a 32 bit integer dynamically defined at the beginning of program operation, which disambiguates who has spoken. Thus if there are three people using the table, there will be three different Client IDs. `Menu` is the complete command list; the system is restricted to matching an utterance against an element on this list. An important feature is that this command list can be dynamically changed at run time to load new command sets from the appropriate Application Mapping file (Figure 4). Thus once a Speech Client has been started, it does not need to be restarted to work with other applications and / or application modes. The `AudioLevel` gives a relative volume; decisions can then be made on whether to ignore speech uttered below a certain threshold. Finally, the client indicates what command it believes the person has said through the `Recognition` field; this is an index into the `Menu` command list. That is, the Recognition index of '1' in this example means that 'label as unit one' has been recognized. Finally, `SpeechEnabled` is true if that speech should be recognized. This could be toggled by an individual (as in Figure 5), or by a floor control policy (discussed later).

3.2 GSI Gesture Clients

Our system uses simple gesture recognition to interpret people's inputs on various touch surfaces. Currently, we use two different table input technologies: the MERL DiamondTouch and the DViT Smart Board. Because each input device has different characteristics that affect what can be recognized, we create specialized clients for each of them (Figure 4, bottom right). However, recognition is delivered to other components of GSI DEMO in a way that hides whether gestures are originating from a MERL Diamond Touch or a DViT Smart Board. Regardless of the technology used, recognition occurs by first training the system (see §GSI Recorder), and then by matching the gesture against this generalized training set. For this later recognition step, we statistically analyze a gesture to produce a set of features, and then compare these features to those defining the different postures produced by GSI Recorder. If a good match is supplied, that posture is recognized and an event is raised. Details specific to each client are described below.

The MERL DiamondTouch Gesture Client. The MERL DiamondTouch, from Mitshubishi Electric Research Laboratories, affords multi-people, multipoint touches [6]. It uses an array of antennas embedded in a touch surface, each which transmits a unique signal. This technology provides an X and Y signal of each user's multi-point contact with the Diamond Touch surface. This produces a projection where the contact gradient provides useful recognition information. For example, Figure 6 graphically illustrates the signals generated by two people: the left bounding box and the top/side signals was generated by one person's 5-finger table touch, while the right was generated by another person's arm on the table. As each user generates a separate signal, the surface distinguishes between their simultaneous touches. For multiple touches by one person, there is ambiguity of which corners of a bounding box are actually selected. Currently, the DiamondTouch surface can handle up to four people simultaneously at 30 frames per second. Computer displays are projected atop this surface, and touch positions are calibrated to the computer display's coordinate system.

Similar to [13], we use this signal information to extract features, and to match these to different whole hand postures for any given instant in time, (e.g., a finger, the side of a hand, a fist). We then detect gestures: these are posture movements over time. Example gestures are two fingers spreading apart, or one hand moving left.

To recognize different gestures from this signal information, we statistically analyze the information to generate a list of 25 unique features (e.g., bounding box size, average signal value, total signal values above a particular threshold, etc). Different values of these features characterize a variety of different possible hand postures. When a hand posture is trained (see §GSI Recorder), the average value of each feature along with a numerical variation estimate is computed and saved. During gesture recognition, the closest matching posture is determined. If it is within the numerical variation limits, a gesture recognition event is generated. If it closely matches two postures or is below a 'confidence' threshold, an unknown gesture event is raised. For ease of use a default gesture set is included that recognizes: one finger, two fingers, five fingers, one hand, one hand side (a chop), one fist and one arm. In practice, this clustering technique has proved to be quite effective across different hand sizes and levels of skin conductivity.

Smart Board DViT Gesture Client. The DViT Smart Board, from Smart Technologies Inc., uses four infrared cameras placed in each corner of the display. This technology is capable of detecting up to two points of contact along with their respective point sizes. Infrared cameras produce a binary signal (on/off) as opposed to

the signal gradient provided by the MERL Diamond Touch. Thus it is somewhat more difficult to recognize particular hand postures (e.g., hand *vs.* chop *vs.* fist). Currently, the technology does not distinguish between different users.

In practice, the point size returned by the DViT can be used to determine the difference between a pen tip, a finger, a hand and a whole arm. However, similarly sized hand postures (e.g., one hand, one fist, one chop) currently cannot reliably be distinguished on the DViT. As well, we cannot identify which person is doing the touch. For now, our DViT Gesture Client recognizes a subset of the Diamond Touch Gesture Engine Events, and all gestures appear as if they are originating from a single person.

3.3 GSI Gesture/Speech Unifier

The next step is to integrate speech and gestural acts. This is done through the GSI Gesture Speech Unifier (Figure 4, Layer 1).

All speech and gesture recognition events are sent to a speech and gesture unifier that is composed of two parts: a single user speech and gesture unifier and a multi-user floor control unifier.

Single user speech and gesture unifier. In our system, speech and gestures can be independent of one another, e.g., when each directly invokes and action. Examples are 'Fly to Boston' to Google Earth, which directs the system to navigate and zoom into a particular city, or a wiping gesture that immediately erases marks underneath the palm. Speech and gesture can also interact as a multimodal command. For example, a person may say 'Create a table' in The Sims, and then touch the spot where the table should be created. In this later case, the gesture/speech unifier must create and deliver a well-formed command to the single-user application from the combined gesture and speech elements.

Our unifier uses Cohen's unification based multimodal integration [12]. If the speech component of a multimodal command is recognized, it must be matched to an appropriate gesture within a certain time threshold, or else the command is treated as a false recognition. In this way it is possible for multimodal speech and gesture to provide more reliable input than speech or gesture alone [10]. The multimodal unification time threshold can be dynamically adjusted by the end user to suit their multimodal preferences. In practice, a threshold of a second or two suffices.

Floor control. Contention results when multiple people interact simultaneously over current single user commercial applications, where each supplies conflicting commands and/or command fragments. For example, if two people try to move an application window simultaneously the window will continuously jump from the hand position of the first user to the hand position of the second user. To mitigate these circumstances, the Gesture/Speech unifier contains various multi-user floor control policies to mediate turn-taking and how people's actions are interleaved with one another.

The output of each single user speech and gesture unifier is delivered to the current multi-user floor control policy module. This module collects and filters the output based on a particular set of rules. The simplest policy is *free for all,* where the system tries to interpret all speech and gestures into well-formed commands. Contention avoidance is left up to the group's social practice. The advantage is that people can interleave their actions, while the disadvantage is that accidental overlap introduces confusion and/or undesired application responses. Other policies balance contention avoidance with social practice. In the case of competing gestures, the *last gesture and last speech wins* policy only listens to the person who had the most recent gesture down event, ignoring the overlapping speech and gesture movements of other users. Our *micro turn-taking* floor control policy adds to this by letting one's gesture acts complete the multimodal speech command of another person. For example, one person can say 'Create a tree', and others can indicate where it should be created. This allows the group to distribute the decision making process to all seated around the table. Other policies (e.g., last gesture and last speech) enforce stricter turn-taking, where others are blocked until the current person has completed a well-formed gesture/speech command.

The organization of the GSI Speech/Gesture Unifier makes it easy to create new floor control policies. Programmers of such policies only deal with high level speech and gesture recognition events rather than low level speech and gesture APIs. They do not need to know if events are generated by the DiamondTouch, the DViT, or even other technologies that could be added in the future.

4. SPEECH/GESTURE BY DEMONSTRATION

For GSI DEMO to work with single user applications, it needs to somehow map speech and gestures into actions understood by that application. That is, it needs to know:

- *the command set*: what commands and arguments should be understood;
- *the speech and gesture commands:* what gestures and speech correspond to those commands and their arguments;
- *the keyboard and mouse acts:* how to invoke application actions when a command is invoked (e.g., mouse down/move/up, menu or palette selections, keyboard entry).

While this information could be hard-wired into the application (as was done in our previously reported versions of our system [11][12]), GSI DEMO lets all this to be configured on the fly by demonstration (Figure 4, layer 2). The idea is that people use the GSI Recorder module to perform a speech / gestural action, and then follow this with the corresponding action on the single user application. The system generalizes the sequence, and packages this up as a macro. When a speech / gesture is recognized, that macro is invoked by the GSI Playback module. While simple, the result is that end-users can quickly repurpose single user applications for multi-user, multimodal digital tabletops. The sections below describe how this is achieved.

4.1 GSI Recorder

The GSI Recorder, shown in Figure 7, lets end-users train the system on how to map speech, postures and gestures to actions understood by the single user application.

However, before the Recorder is started, it must have two things. First, speech recognition needs a speech profile customized for each person; we use the Microsoft SAPI system to generate these individual profiles. Second, a set of possible postures is needed, as people's gestures will be matched to this set. To train a posture, a person places a posture (e.g., a fist) on the surface, and then moves it around to various positions and angles over the surface for a short time. Instances of this posture are generated 30 times a second. As mentioned previously, statistics are performed over

the raw signal or point information defining each instance to convert them into a number of features (e.g., the average signal in the bounding box surrounding the posture). Features across these instances are them generalized using a univariate Gaussian clustering algorithm to produce a description of the posture, saved to a file, and later used by the Recorder.

Training. Three different types of speech/gesture to keyboard/mouse mappings are supported by the Recorder: *continuous gesture commands* (e.g., a one hand pan is remapped onto a mouse drag), *discrete speech commands* (e.g., "fly to Boston" is remapped onto a palette selection and text entry sequence) and *multimodal speech and gesture commands* (e.g., "move here [pointing gesture]" is remapped onto a command sequence and mouse click).

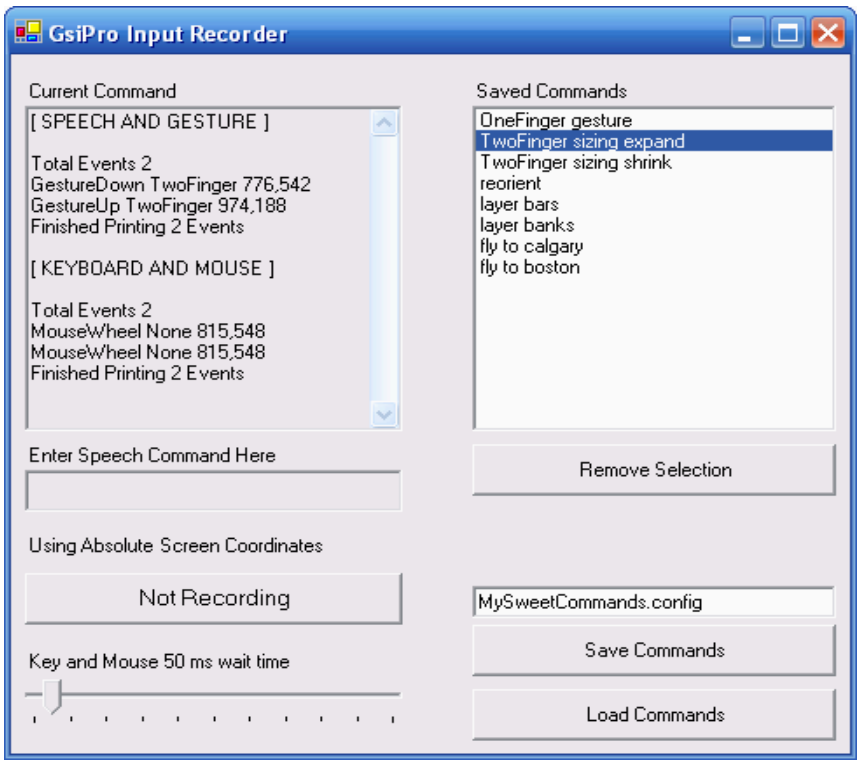

Figure 7. The GSI: Recorder GUI

Continuous gesture commands are trained by saying "Computer, when I do [gesture] you do [mouse sequence] okay". To break this down, GSI Recorder will attempt to recognize a hand gesture after the person says "Computer, when I do". If it does, it provides visual and feedback (Figure 7 top left). The person continues by saying "you do" and then performs the keyboard/mouse sequence over the single user commercial application. Because the application is live, the person receives instantaneous feedback about the status of the mouse sequence; abstracted actions are also displayed graphically (Figure 7 top left). The person says 'okay' to end the keyboard and mouse sequence. After the gesture is recorded, people can instantly test what was learnt by performing the appropriate gesture on the tabletop. If it is wrong, they delete it (the 'Remove Selection' button) and try again.

Discrete speech commands are trained similarly. The person says "Computer, when I say [speech command] you do [keyboard and mouse sequence] okay". Speech commands are typed in rather than spoken; later, the speech recognizer will parse spoken speech – regardless of who said it – to see if it matches the textual string. This is important, as it means that one person can train the system and others can use it. As before, discrete commands can be played back by saying the recently recorded speech command.

Multimodal speech and gesture mappings are created by saying "Computer, when I say [speech] [gesture] you do [keyboard and mouse sequence] okay". Gesture can also precede speech if desired. Again, speech commands can be quickly played back by performing the appropriate sequence of events.

Each successful mapping is recorded to a list box (Figure 7, right) that stores all successful commands. Clicking on a command reveals the recorded gesture and its corresponding keyboard and mouse sequence (Figure 7, left). The 'Save Commands' button stores the entire command set and its mapping between speech/gesture and keyboard/mouse actions to an Application Mapping File (Figure 4). The GSI Player will later load and use this file.

Generalization. Under the covers, GSI Recorder uses the Microsoft Windows Hook API to listen to the keyboard and mouse events of all applications while the end user interacts with the commercial application. However, it would be inappropriate to just record and playback this literal stream. Some generalization is needed.

The main generalization is how recognized gestures are translated into mouse events. For each gesture, the Recorder records its 'Gesture Down' and 'Gesture Up' event (along with its coordinates, posture, bounding box, and so on). It does *not* record all intervening 'Gesture Move' data. During later replay, it matches gesture to mouse coordinates by first taking the center of the posture as the mouse down, then the intermediate gesture points as the mouse move, and then the final gesture up point as the mouse up. Of course, this generalization means that complex gestures cannot be recognized.

We also found that there are times when it is inappropriate for the direction of a continuous gesture command to directly map onto the click coordinates of the corresponding mouse command. When the middle mouse button in some applications is held down to scroll, moving the mouse down (closer to the user) advances the document away from the user. In a gestural interface, the document would appear to be moving in the opposite direction than expected. GSIRecorder recognizes these cases. If a person specifies a gesture in one direction, but then moves the mouse in the opposite direction, it will invert the input appropriately.

Another generalization is that the literal timing information in keyboard and mouse moves is discarded. During replay, this has the advantage that long sequences of keyboard/mice actions are

replayed quickly. As well, if sequences raise and then hide menus, tool palettes or dialog boxes, these pop-up screen elements may appear as a brief flash, or not at all. This gives the end users the illusion that the executed keyboard and mouse sequence is invoked as a single command. For example, the "layer bars" command in Google Earth is really a complicated sequence of opening the layer menu, toggling the bars option and closing the layer menu. This command appears as a brief 0.1 second flicker, with the end result of bars being layered on the digital map. However, we also recognize that some applications do need delays between events (e.g., time for a menu to fade away); in these cases, people can use the 'Key and Mouse' trackbar (Figure 7, bottom) to specify particular wait times.

Finally, another generalization considers the screen coordinates of mouse actions. Mouse events are normally received in absolute screen coordinates. If the application window is moved, or if interaction on a popup window is required, or if the screen resolution changes (thus shifting how some windows are arranged on the screen), the mapping will not work. One partial solution lets the user specify that actions are relative to a particular window by saying: "Computer, relative to window [select window]". GSIRecorder then uses the GroupLab.WidgetTap toolkit [7] to find the string uniquely identifying that window. When the command is later executed, GSIPlayer will then search for the appropriate window and then make all coordinates relative to its top left corner.

4.2 GSI Player

GSI Player loads the set of speech/gesture to keyboard/mouse mappings from the Application Mapping file (Figure 4), which is displayed in the Player's interface (Figure 8, top). Different files can be loaded for different applications, or even for different modes within an application. For example, we generated different mapping configurations for two variations of GoogleEarth, for The Sims, and for Warcraft III. When a file is loaded, the appropriate speech command set is sent to each GSI Speech Client, and the GSI Gesture/Speech Unifier is set to monitor appropriate speech and gesture commands as specified in the file. To the end user it appears as if the system now seamlessly accepts speech and gesture multimodal commands from that command set.

GSI Player lets people set a variety of other options (lower half of Figure 8):

- what floor control policy should be used (via the 'Multiuser Floor Control Policy' dropdown list);
- maximum time that must pass between speech and gestural acts if they are to be unified;
- whether a speech command prefix is to be used to enable utterance recognition, and if so what that should be (done by altering the text in the 'Speech Prefix' textbox
- toggle auditory feedback that happens when a command is recognized ('Play Sound on Speech Recognition' box).

GSI Player also automatically supports true multi-user annotation over the single user application through its 'Scratch Pad' speech command. Using the GroupLab.Collabrary [1], the GSI Player captures an image of the primary screen, and then overlays this image on top of the commercial application. To the end user, it appears as if the application has 'frozen'. Multiple users can then simultaneously annotate with different colors per user using a single finger. Any posture larger than a single finger is treated as an erasing gesture, where erasure restores the underlying background image. Saying 'Scratch Pad' a second time returns to the original single user application. The collaborators see the marks disappear, and the application comes alive again.

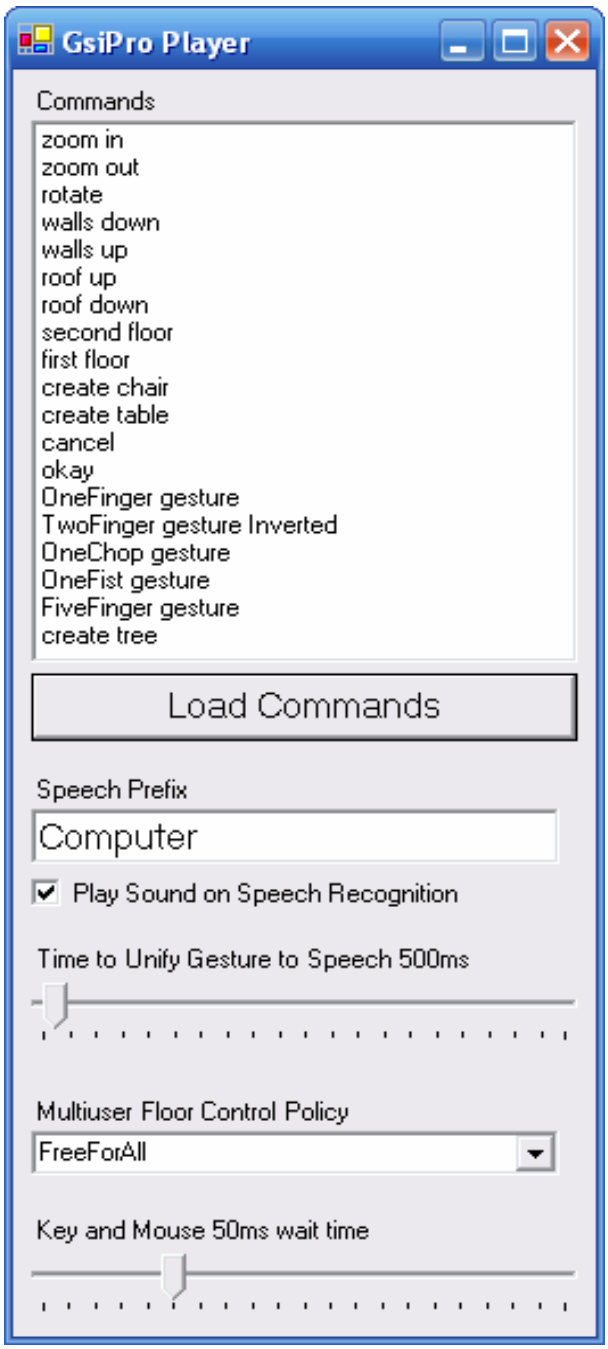

Figure 8. The GSI Player GUI

5. DISCUSSION

We prototyped four gesture and speech wrappers over existing single user applications. Three of these applications (Google Earth, Warcraft III, and The Sims) had been previously hand

coded as custom multimodal wrappers [11][12], a process that often took weeks to create and that were difficult to maintain and alter. By way of contrast, all three wrappers took under an hour to do by demonstration. As well, we could train by demonstration all of the interactions used in our previous papers [11][12], where speech and gestures were recognized and the corresponding single user commands invoked. Finally, programming by demonstration meant we could try out gestures, see how they worked, and discard them for other alternatives if desired.

A new example we instrumented is Virtual Knee Surgery, an educational tutorial produced by EdHeads.org of the steps involved with a knee surgery operation. This tutorial puts people in the role of a surgeon who must select the appropriate tools to perform the operation. By default, virtual surgery displays a list of tools on the bottom of the screen (closest to the upright user on the table). The user must click on one of the tools in order to select it. While this metaphor works well for a single user over an upright display, the fact that the buttons disappear once a selection has been made makes the action private and unapparent to other members around the table. For this reason, others cannot double check one's actions and provide assistance when necessary just like they would in a real surgery. To facilitate double checking and assistance, we implemented tool selection as a set of verbal alouds in Virtual Surgery, and actions as a series of gestures. Just as a doctor would ask a nurse for a scalpel, participants can ask the computer to provide them with the appropriate tool and then do the action via an understandable gesture. This verbal aloud serves a double function as a computer command and as an awareness tool to inform collaborators about the actions that they are about to take. Different collaborators could also try out different surgical actions through gestures.

Limitations. GSI DEMO is not perfect. First, it is limited by the single user applications it wraps. While some multi-user interaction can be performed by interleaving individual actions, the underlying application is oblivious to this fact. As well, the mechanics of invoking certain application actions have side-effects, e.g., if a speech command invokes a graphical menu selection, the menu may briefly flash on the display.

Second, GSI DEMO is a crude programming by demonstration system. It makes many simplifying assumptions on how gestures should be mapped to application actions, and makes no attempt to learn across examples as done in other programming by example systems [5] and even in other gesture recognition systems [2]. For example, it assumes that there will be a simple mapping between a gesture (e.g., a whole hand move) and a mouse click/drag. While this makes it easy to (say) map a hand move to a mouse panning action, mapping a two finger touch to a double tap is not possible. Similarly, complex gestural shapes (e.g., drawing a flower) are not recognized. Of course, an obvious step in this work is to incorporate techniques forwarded by the machine learning and programming by demonstration communities [5][2].

Third, multimodal mappings are currently limited. Only a simple speech + one gesture (e.g., finger, hand) can be mapped to a series of keyboard commands followed by a mouse click. Mouse click coordinates are remapped to the center position of the gesture. This means that complicated multimodal commands that (say) involves clicking a menu before specifying a location would fail.

Fourth, macros and mappings can be deleted and reconstructed, but not edited. This means that long sequences would have to be recreated, even though a small edit would fix it. Again, methods exist to edit macros, e.g., Halbert's early SmallStar system [5].

In spite of these technical limitations, we were able to define a reasonable working set of rich, natural actions atop our single user applications. Perhaps this is because the actions people want to do when collaborating over a surface are simple and obvious.

6. CONCLUSION

We contributed GSI DEMO, a tool designed to let digital tabletop users create their own multi-user multimodal gesture and speech wrappers around existing single-user commercial applications. GSI DEMO circumvents the tedious work needed to build gesture and speech wrappers from scratch. Training the computer is as easy as saying "Computer, when I do [gesture action] you do [mouse action]" or "Computer, when I say [speech action] you do [mouse and keyboard macro]". This gesture and speech wrapper infrastructure allows end users to focus on the design of their wrappers rather than underlying plumbing.

In the future we hope to advance the generalization capabilities of GSI DEMO such that it would be able to understand and invoke the semantic meaning of one's actions (e.g., saving a file) rather than the primitive and perhaps erroneous mouse and key combinations. Better generalization would allow the computer to understand what people are trying to do rather than how they are doing it [5][2]. We also recognize that GSI DEMO currently supports only simple speech and gesture mappings. Yet we foresee a new version that could handle more complex gesture to mouse actions, and that this would further support end users' creative freedom in designing multi user multimodal speech and gesture wrappers around existing single user commercial applications.

Acknowledgements. Thanks to Sheelagh Carpendale, Kathryn Elliot, Clifton Forlines, Cheng Guo, Gregor McEwan, Stephanie Smale, Nelson Wong, Daniel Wigdor, Jim Young and our sponsors: Alberta Ingenuity, iCORE, NSERC, MERL and Smart Technologies Inc.

7. REFERENCES

[1] Boyle, M. and Greenberg, S. Rapidly Prototyping Multimedia Groupware. *Proc Distributed Multimedia Systems* (DMS'05), Knowledge Systems Institute, 2005.

[2] Cao, X. and Balakrishnan, R. Evaluation of an online adaptive gesture interface with command prediction. *Proc Graphics Interface*, 2005. 187-194.

[3] Cohen, P.R., Coulston, R. and Krout, K., Multimodal interaction during multiparty dialogues: Initial results. *Proc IEEE Int'l Conf. Multimodal Interfaces*, 2002, 448-452.

[4] Cohen, P.R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L. and Clow, J., QuickSet: Multimodal interaction for distributed applications. *Proc. ACM Multimedia*, 1997, 31-40.

[5] Cypher, A. *Watch What I Do: Programming by Demonstration*. MIT Press, 1993.

[6] Dietz, P.H., Leigh, D.L., DiamondTouch: A Multi-User Touch Technology, *Proc ACM UIST,* 2001. 219-226

[7] Greenberg, S. and Boyle, M. Customizable physical interfaces for interacting with conventional applications. *Proc. ACM UIST Conference,* 2002, 31-40.

[8] Gutwin, C. and Greenberg, S. Design for individuals, design for groups: Tradeoffs between power and workspace awareness. *Proc ACM CSCW,* 1998, 207-216
[9] McGee, D.R. and Cohen, P.R., Creating tangible interfaces by augmenting physical objects with multimodal language. *Proc ACM Conf. Intelligent User Interfaces*, 2001, 113-119.
[10] Oviatt, S. L. Ten myths of multimodal interaction, *Comm. ACM*, 42(11), 1999, 74-81.
[11] Tse, E., Shen, C., Greenberg, S. and Forlines, C. Enabling Interaction with Single User Applications through Speech and Gestures on a Multi-User Tabletop. *Proc. AVI 2006*.
[12] Tse, E., Greenberg, S., Shen, C. and Forlines, C. (2006) Multimodal Multiplayer Tabletop Gaming. *Proc. Workshop on Pervasive Games* 2006.
[13] Wu, M., Shen, C., Ryall, K., Forlines, C., Balakrishnan, R., Gesture Registration, Relaxation, and Reuse for Multi-Point Direct-Touch Surfaces, *Proc 1st IEEE Int'l Workshop on Horizontal Interactive Human-Computer Systems (TableTop2006)*. 183-190.