6 System architecture

Cursive is an application for interactively controlling the animation of VRML avatars. It uses the pen interaction technique described in *Chapter 3 - Interaction technique*. It is used in conjunction with so-called "browser only multi-user" virtual environments to send animation commands to avatar drones across the internet. Written without the use of any vrml browser or virtual world software API's, Cursive communicates directly with the avatar instances. Because it bypasses the virtual environment system communication protocols, other users logged into the same virtual world can view the avatar's animation without modifying their existing software configuration. This application benefits avatar designers who want to display animation behaviors to a public audience without committing themselves to a particular virtual world system or forcing the audience to adopt specialized software.

6.1 Cursive

Cursive is a distributed application that works with avatars in a VRML (Virtual Reality Modeling Language) based virtual world. The gesture command portion of the software runs on the user's local host. It provides the user interface, accepts the user's pen gesture input, generates the avatar gesture animation commands, and sends the commands to the avatar via a socket connection.

The animation portion runs on every host, local and remote, where instances of the user's avatar appear. This portion runs inside of a VRML browser and consists of Java code that is part of the VRML avatar's script nodes.As it receives the gesture commands from the network, it animates the avatar.

A screen shot of the Cursive window alongside a VRML browser is shown in Figure 6-1.

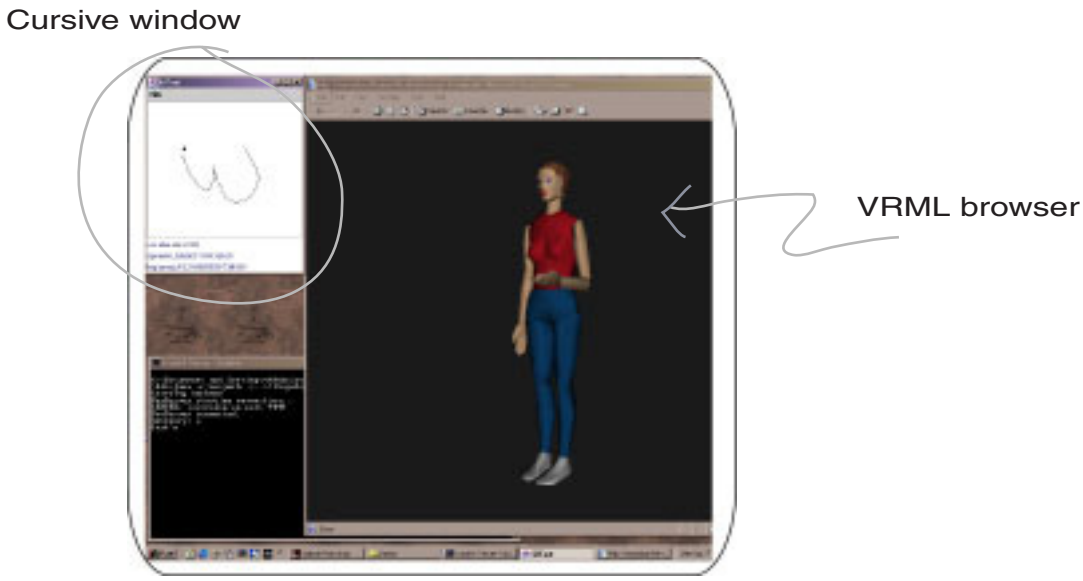

Figure 6-1. Cursive window next to a VRML world browser.

The Cursive window appears in the upper left hand corner of the screen. The VRML browser displaying the user's avatar covers the right hand side of the screen.

6.1.1 Portability

This application provides a testbed for the development of our interaction technique. We intend to experiment with it in public virtual worlds. To maximize the number of potential viewers of our gesturing avatar, we designed Cursive to be independent of any specific virtual world software. The files describing the avatar object (both its geometry and behaviors) consist of standard VRML 2.0 files and some Java code associated with the VRML script nodes.

The architecture is designed so that the avatar's animation behaviors are not dependent on the specific virtual world (VW) software being used as long as the world is viewable using a standard VRML browser. Neither the VW server nor the client are modified to support these behaviors. When the user logs into a virtual world, the control enhanced avatar is uploaded just like any other avatar. As the user navigates the world and animates the avatar, the avatar's movement can, in principle[1], be seen by any users who are viewing the virtual world. The viewers are unaware of the source of the avatar's animation.

6.1.2 Interest

The design of this application is of interest to any VW content developers who want to implement portable interactive behaviors for avatars and other distributed objects. An application such as this will allow designers to view and test designs for their objects without committing to a particular virtual environment platform. It allows these objects to be dropped into any VRML based virtual world.

6.2 Virtual world application model

Normally, a user runs proprietary software, called a VW client, to log into a virtual world, communicate with other users, and navigate their avatar around the world. Cursive runs alongside this software without interfering with the client's avatar navigation controls or any of its other functions. To describe how Cursive is able to control an avatar in this execution environment, we first review the workings of a generic multi-user virtual world application.

[1]In practice, not all VRML features are implemented by all VRML browsers. The functionality of the VRML script node Java interface varies widely. The application has been used successfully in Cortona and Cosmo running in Microsoft Explorer 5.0.

For illustrative purposes, we describe a client-server VW application although Cursive's architecture should allow it to work with a peer-to-peer application as well.

A conceptual model of the example VW application is shown in Figure 6-2. The virtual

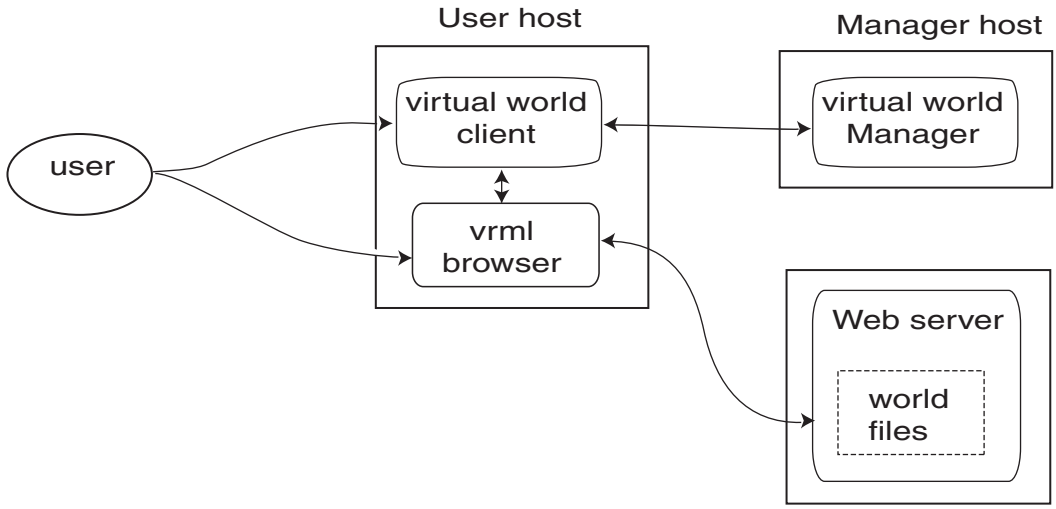

Figure 6-2. Conceptual model of multi-user virtual world application

world clients we have used run as applets within a web browser, and the VRML browsers are plug-ins to the web browser. Conceptually, we can think of the VW client and VMRL browser as running side by side and communicating via their own application interface.

6.2.1 Virtual world manager

The virtual world manager serves the world objects (the VRML files and script code which make up the world), and manages the state of the world. The state includes the logins and logouts of users and the position and orientation of the each user's avatar within the world. Whenever the state of the world changes through the actions of one of the users, the change gets sent to the VW manager to be propagated to the rest of the VW clients. The manager also caches the URL's of any user-uploaded objects, including avatars.

6.2.2 Virtual world client

The user logs into the manager using VW client software on their own host. The client sends and receives updates to the world state. As the user moves their avatar, known as the *pilot* avatar, around the world, the client detects these motion events and sends the updated coordinates of the avatar to the virtual world server. When the other clients receive the updates, they update their local instances of the avatar, known as *drones*[2], by sending motion events to the VRML browser. The client communicates with the virtual world manager using an internet protocol such as TCP or UDP.

6.2.3 VRML browser

A VRML browser manages the local state of the world, and renders the worlds including any avatars in it. Although the user logs into the world using the VW client, they interact with the world through the VRML browser. In fact, all changes to the world are originated in the VRML browser by the user. The VRML browser has its own user interface separate from the virtual world client's browser.

User actions that affect the world initially update the local state. These actions include navigating the pilot avatar around the world as well as manipulating other pilot objects. For instance, clicking on a door object may cause the door to open. After the update, the change events are made available to the VW client. The VRML browser also receives change events, affecting drone objects, from the VW client. These are events that result from the actions of remote users at other VRML browsers.

---

[2]The terminology *pilot* and *drone* are borrowed from *Living Worlds*, a proposed standard for distributed interaction in VRML virtual worlds. "Pilot" refers to the avatar instance, or any object instance, that originates a behavior. "Drones" are instances that reflect the behavior of the pilot.

6.3 Cursive communication

Cursive is designed to work with a VRML BOMU (Browser Only Multi-User[3]) Virtual World, meaning that only a standard VRML browser is required for viewing the world. We take advantage of the fact that the VRML browser can execute code that extends VRML object behaviors. We use this functionality to set up communication between the user and the avatar. This allows Cursive to communicate with the avatar through the VRML browser and without having to use the protocols of the virtual world manager and client. It also allows Cursive to communicate with the avatar running on the machines of other viewers. In the rest of this description, we will differentiate among types of users as follows.

User. Refers to a human who is running Cursive, viewing an avatar through the vrml browser, or running a virtual world client.

Driver. Refers to a Cursive user who is controlling an avatar.

Viewer. A user who is viewing a virtual world, and in particular, who is viewing the gesturing avatar. Note that the driver is also a viewer.

[3]BOMU is also part of the Living Worlds specification.

Initially, before the Cursive driver even logs into the virtual world, they are running the VW client, the Cursive application and a web server on their machine as shown in Figure 6-3.

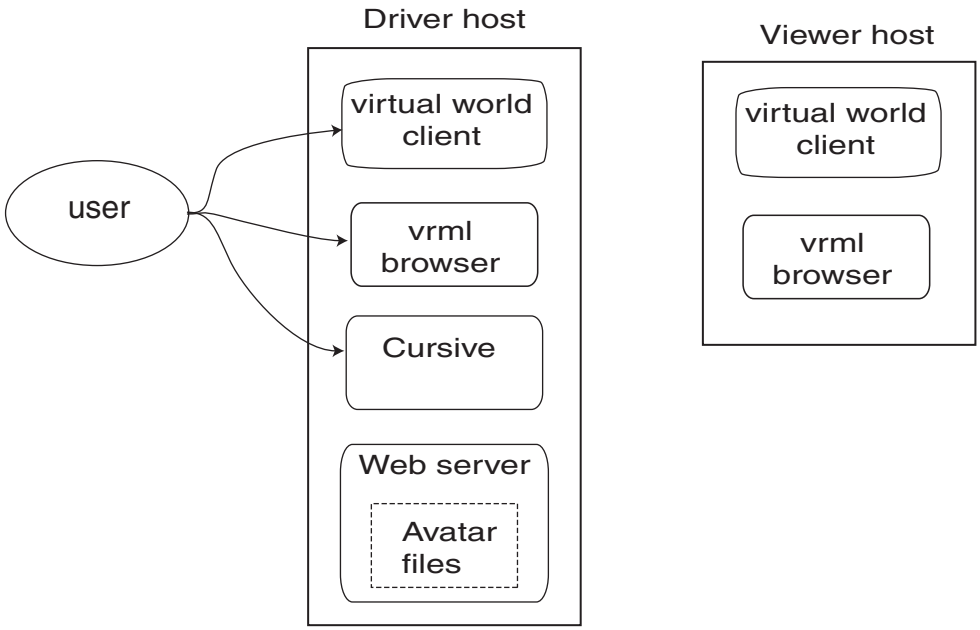

Figure 6-3. State of system before user log's in to virtual world.

The local web server will be used to serve the user's avatar files.

Recall that in the standard model, the user controls their avatar using the VRML browser interface, these changes are applied to the local state, and then are sent to the virtual world client. Cursive provides its own user interface for controlling the avatar gesture and communicates directly with the avatar through VRML script nodes, bypassing the VW client and server. The steps involved in setting up the network communication within the Cursive application are described in the next few sections.

6.3.1 Updating virtual world state

The Cursive user logs into the virtual world using the VW client and supplies the URL for their avatar. The client communicates this event to the VW manager, usually through a

TCP connection. Once the login is registered, the VW manager responds by sending back the current world state and URL's for the world model and for the other avatars in the world. All other user's are informed of the new login and they receive the URL for the Cursive avatar. This updating is shown in Figure 6-4.

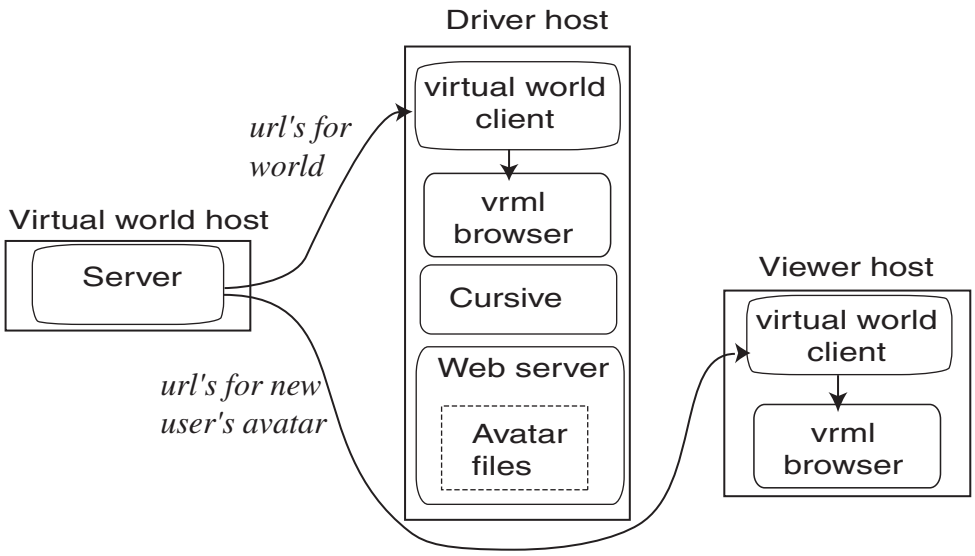

Figure 6-4. Virtual world server updates virtual world clients.

The actual VRML files are downloaded by the VRML browsers from web server hosts using HTTP. Figure 6-5 shows the VRML browsers downloading the Cursive avatar.

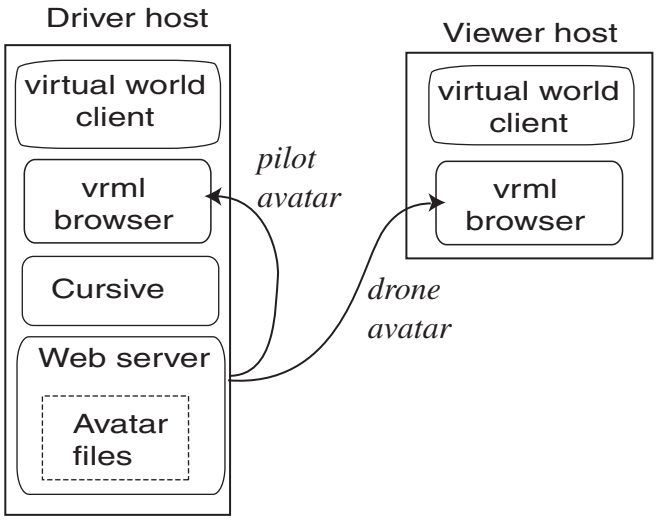

Figure 6-5. Virtual world browsers download the new user's avatar.

6.3.2 Gesture command communication

Among the files that make up the Cursive avatar are files containing the Java code for communicating with Cursive and for animating the avatar. This animator code is executed within the VRML browser as shown in Figure 6-6. (Actually, it is executed by the web browser that is hosting the VRML browser. For simplicity, we are eliding the presence of the web browser.)

To receive the gesture commands, the Java code opens a network socket connection to Cursive. In general, a web browser's security manager will not allow external code to open a network connection. In this case the socket connection is allowed because Cursive is running

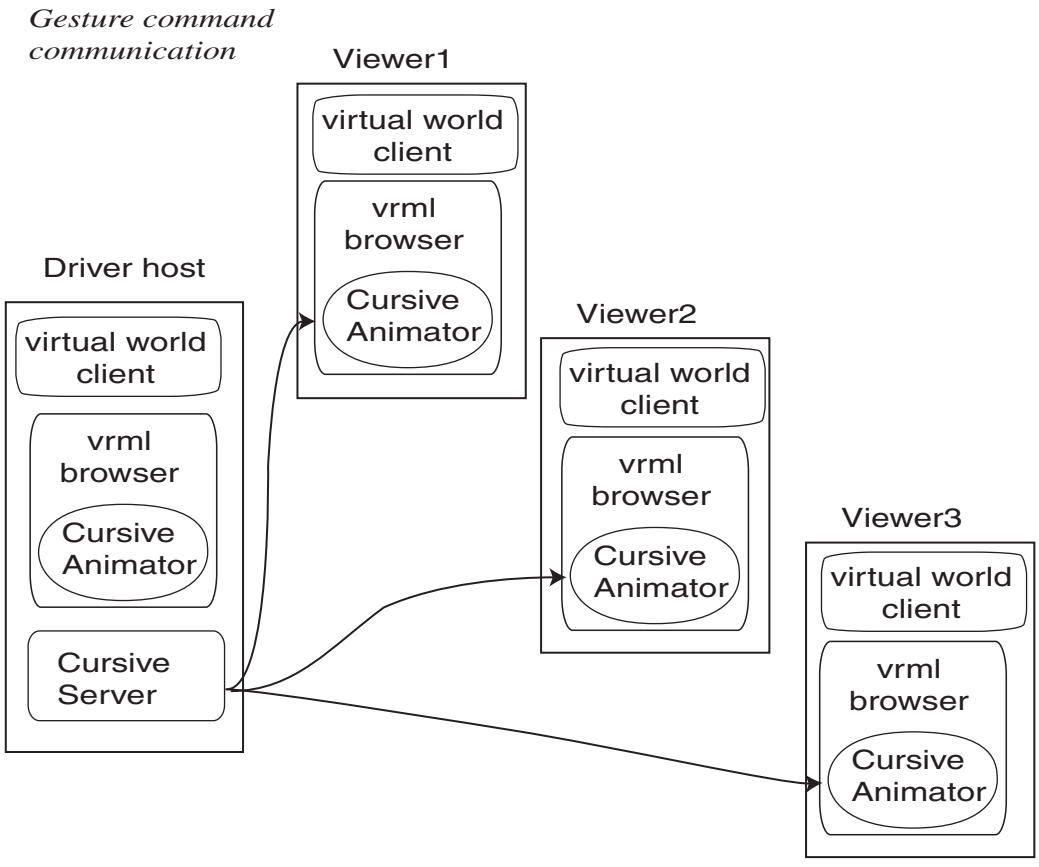

Figure 6-6. Gesture commands sent to all copies of user's avatar.

on the same host as the web server that served the code. This is the reason that the driver must serve their avatar's files from their local host.

From then on, the user can control their avatar through Cursive. The gesture commands are sent to all avatar copies via these sockets as shown in Figure 6-6. Unlike the standard model, there are no avatar drones. Recall that normally pilot objects are manipulated through the VRML browser user interface and that these events then get passed to the client whereas drones receive events from their own client.In the Cursive model, none of the avatar instances is privileged in this way since all receive their commands through a socket connection.

6.4 Cursive functional modules

Cursive is the implementation of the our interaction technique for controlling expressive avatar gesture using pen gesture, as described in *Chapter 3 - Interaction technique*. The code is written entirely in Java. A schematic for the modules is shown in Figure 6-7.

6.4.1 Pen UI and pen gesture feature analyzer

The Pen user interface puts up a window for the user to write into. Digital ink from the pen gesture is then passed on to a handwriting feature analyzer. Some of the features are used by the character recognizer to determine the identity of the pen gesture. Other style features are passed directly to the avatar gesture modulator to be mapped to motion modulation parameters.

The code for the pen user interface is a modification of the Quill program, a pen gesture training and evaluation application developed by Long at U.C.Berkeley [68]. Quill implements the feature-based pen gesture recognition algorithm developed by Rubine[93].

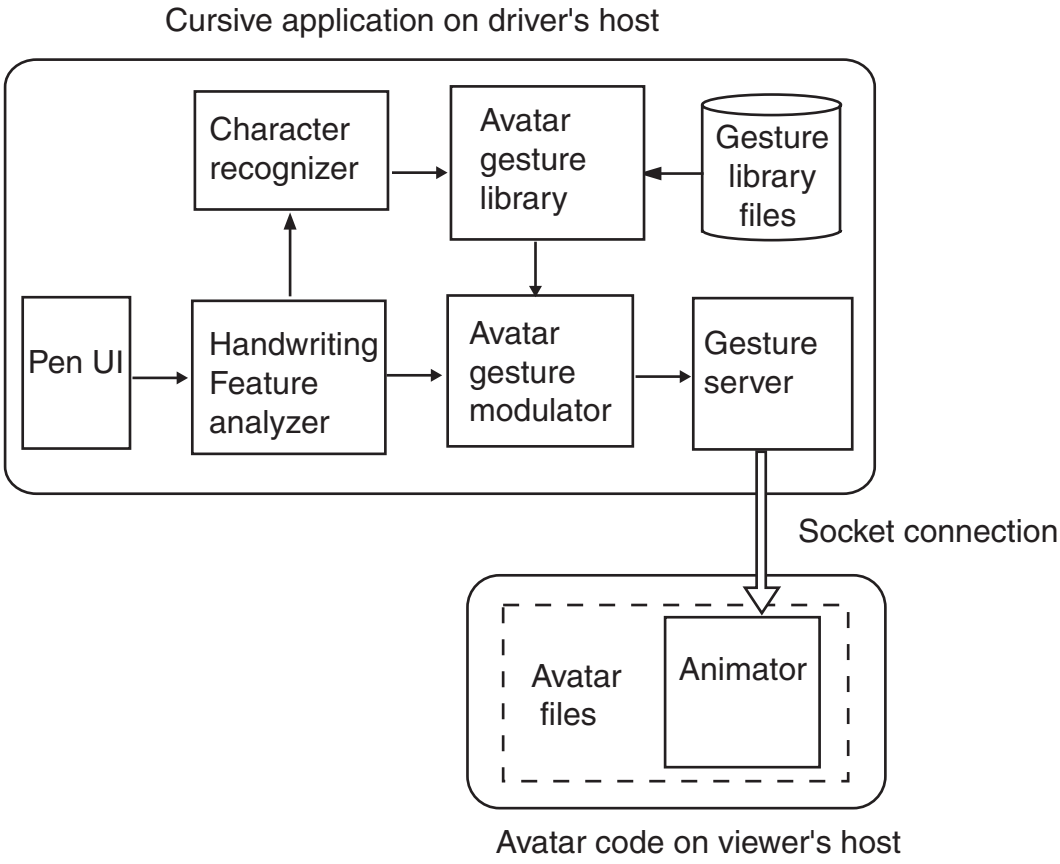

Figure 6-7. Cursive functional modules.

6.4.2 Gesture generation

At runtime, the gesture library files are read from disk into the avatar gesture library and become available online. The pen gesture identity, computed by the character recognizer, selects a particular avatar gesture type from the gesture library as shown in Figure 6-8a. The data for the selected gesture is passed to the avatar gesture modulator along with a vector of handwriting style features. The avatar gesture modulator synthesizes a specific gesture motion that is specified by the style features. Recall that we synthesize gesture by interpolation from motion samples as shown in Figure 6-8b, and that the style features are mapped to the interpolation parameters.

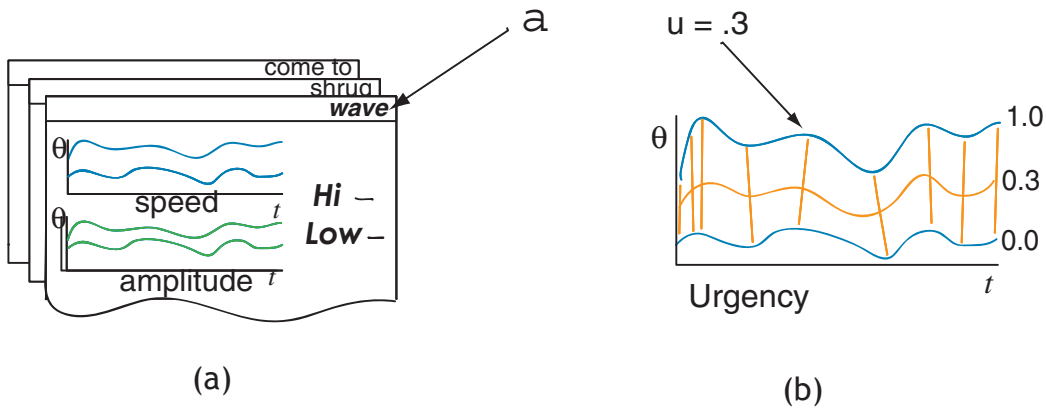

Figure 6-8. (a) Letter identity selects particular gesture; (b) features modulate motion.

6.4.3 Gesture command server

The Gesture server transmits the avatar gesture motion commands to the avatar over a socket connection. When Cursive runs, the server waits for socket connection requests from copies of the user's avatar that have been loaded into a VRML browser. Once a connection is established, the server sends avatar gesture commands to the avatar as long as the avatar maintains the connection.

An avatar gesture command consists of a series of key frames and time stamps indicating the relative timing of the key frame within the gesture animation. The gesture command protocol consists of two kinds of packets: timestamp packets and joint value packets. At the beginning of a key frame, the servers sends a timestamp packet indicating the time of the key frame relative to the beginning of the gesture. The timestamp packet also indicates the number of joints that will be described in the key frame. A packet with timestamp *0* indicates that a new gesture is beginning. Then the key frame values are sent out as a series of joint value packets, one packet for each joint that is part of the animation.

Time zero for the gesture begins from the time the letter is recognized. At present, the letter is recognized only after the pen is lifted. However, in a continuous recognizer, the pen

would not have to be lifted for the letter to be recognized. Rubine's pen gesture recognition algorithm also supports continuous recognition, but we have not taken advantage of this feature as it was not implemented in Quill.

6.4.4 Gesture animation

The gesture animation code is part of the VRML script nodes belonging to the avatar model files. The key frames for a gesture are streamed to the avatar as soon as they are available. The animation code is executed inside of a VRML browser and is responsible for playing back the gesture with the correct timing.

The key frames are stored in a queue, and each frame is dequeued at the time it should appear in the animation. When a timestamp packet comes in with value *0*, the current time is noted as the real start time of the avatar gesture. As each successive timestamp packet is received, the key frame associated with that time stamp is marked with the real time at which it should be animated, that is, the start time plus the time value of the timestamp. Since commands can be sent faster than the gesture itself can be performed, more than one gesture may be on the queue at the same time. In this case, the gestures are just performed successively.

In general, the avatar animation update rate that will differ from the frame rate of the gesture commands. For instance, the key frames for a gesture may be drawn from joint angle trajectories sampled at 20 Hz while the avatar may be animated at a rate of 12 Hz. Because of this difference, the angles used to update the avatar animation will usually be interpolations of angles specified in the command gesture key frames. The joint rotations are represented using quaternions, so quaternion interpolation, as described in Section 4.3.3, is used to calculate the interpolated frames. However, other interpolation methods, such as spline interpolation, may provide better animation.

6.5 Summary

Cursive demonstrates an application in which pen gestures are used to control avatar gestures. The architecture of this application was designed so that expressive avatars can be controlled in a virtual world without having to modify the virtual world infrastructure. As a consequence of this design, it should be possible to "drop" expressively animated avatars into existing avatar worlds.