Distributed Collaboration - Assignment 4: Creating, Detecting and Overcoming Jitter

Date Assigned: October 29, 2012

Completion Date: (Friday) Nov 9, 2012

Objectives:

- Introduce jitter in your application
- Detect and overcome jitter.

In your previous assignment, you added delays to the communication links. In this assignment you will also add jitter to the links, and implement one or more mechanisms to overcome the jitter problems that arise from telepointer movements and possibly other messages.

Jitter occurs when messages from a source to a delay have varying rather than constant delays. The common reason for jitter is queuing in routers: a router may release together a series of messages that arrived at different times. If the sender time stamps the messages, then a receiver can detect jitter by comparing the sending time stamps with the receive times. If the difference between the receive times is substantially smaller or greater (based on some threshold) than the difference between the sending times, then there is jitter. There are many ways a receiver can overcome from jitter. One is to add additional delays to the messages to make sure that if the send time of M2 was T units greater than the send time of message M1, then the processing time of M2 will also be at least T units greater than the processing time of M1. The processing time of a message is time when the message is delivered to the model. A variation of this technique is for the receiver to buffer the incoming messages instead of playing them immediately. By doing so, it can make sure that that M2 is processed exactly T units after M1.

Another technique , consistent with the idea of buffering, and meant for telepointers, is to show *telepointer trails* when jitter is detected. This means that a receiver processes a series of (buffered) telepointer positions, P^1 … P^N, not by making the telepointer move to each of these positions, but instead, by drawing a line between P^i and P^{i+1}, for all 1 <= i <= n-1, and making the telepointer move to position P^N. A telepointer trail is erased when the telepointer comes to rest or when a new series of message is played. A whole trail does not have to be erased at once; instead, it can be gradually erased. In fact, each new trail segment addition can result in erasing of the oldest previous trail segment. The number of segments in a trail can get very large. You can filter out telepointer movements if they occur less than 100 ms apart. In addition you can use a curve model to smoothly fit a series of points. The uigen class ACurveModel defines a model that displayed as curve by AWT.

In this assignment you should implement the telepointer trail technique for telepointers. For extra credit, you can also implement one or both of the other more general techniques for non telepointer messages, and provide a way to dynamically switch between the various techniques.

You are free to choose the exact architecture and algorithm for addressing jitter. Here is a suggested architecture and algorithm, which I have not actually implemented, and so may have flaws.

An in coupler will not deliver a message it gets from an out coupler directly to the model. You will create a new in-coupler that delays or adapt the message, using additional jitter-related modules, before it delivers the message to the original in coupler. Moreover, to show telepointer trails, the telepointer model(s) will have to be extended to create the connected lines. Finally, an out coupler will not send messages directly to the in couplers. Instead, you will create a new out coupler that adds jitter and then delivers the message to the original out coupler.

To add both delay and jitter at the sending site, create two threads, a delay thread and a jitter-addition thread. The out coupler time stamps the property changes it receives and queues them for the delay thread. The delays thread delays the messages before queuing them for the jitter-addition thread. These delays could be constant or have a mean and variance, and could depend on the destination. The jitter-addition thread models a network router by releasing all queued packets, sleeping for some time, releasing all queued messages, and so forth. The sleep time can be constant or have a delay and variance. For this assignment, it is sufficient to add only jitter.

At the receiving site, a new out coupler will queue elements for the jitter-subtraction thread. The jitter-subtraction thread runs in the normal and jitter-recovery mode. It starts in the normal mode. It goes into the jitter-recovery mode when the first message it receives in the normal mode has jitter. It goes from the buffered mode into the normal mode when the first message it receives in this mode does not have jitter. A message has no jitter if its receipt time – the receipt time of the previous message is about the same as the send time of the message – the send time of the previous message.

In the normal model, it simply delivers each message in the queue to the previous out coupler. In the jitter-recovery mode, it does the following. It sleeps for some time T which is close to the frame rate in a video feed (30-100 ms), submits messages to the original out coupler that erase the last trail, removed all buffered messages, converts the buffered messages into a new telepointer trail, submits messages to the original out coupler to add this sleeps for time T, and continues this process until it encounters the first no- jitter message or there are no buffered messages. In either case, it sets the mode to normal. If there are no buffered messages, it simply waits for the next message queue.

You should make your jitter recovery module independent of the algorithm for creating trails. The module should take in its constructor:

1. an object for converting messages it receives from the new out coupler into messages that represents a telepointer trail.
2. an object for converting messages that represent a telepointer trail into a sequence of messages that erase the trail

If you are not doing extra credit, you should add jitter to and remove jitter from only telepointer movements.

Submission

Create and post a video showing the local and remote feedback to a telepointer movement :

- With no jitter and jitter recovery
- With jitter and no jitter recovery
- With jitter and jitter recovery

In addition, give me a printed document outlining which of the jitter recovery algorithms you have implemented and pseudo code for the algorithms.