Implementing a multi-media conferencing tool - Experiences

Jack Jansen

Centrum voor Wiskunde en Informatica

Kruislaan 413

1098 SJ Amsterdam, the Netherlands

jack@cwi.nl

Overview

- Introduction:
  - history,
  - goals,
  - overview,
  - user interface
- Implementation tools
- Application structure
- Conference control
- Audio
- Other media
- Conclusions

History and goals

The project was started about 2 years ago. The initial goal was more of a meta-goal: gaining experience with multi-media conferencing tools, both from an implementation standpoint and (to a lesser extent) from a user standpoint.

Current goals include:

- designing a framework in which there is a clear separation between conference control and media handling,
- looking it issues related to tightly controlled sessions.

The implementation has recently undergone a complete redesign (after the previous version had gone through a few major rewrites). For each of the subjects treated we will look at the current design and at previous designs and point out what the reasons for redesigning were.

Overview

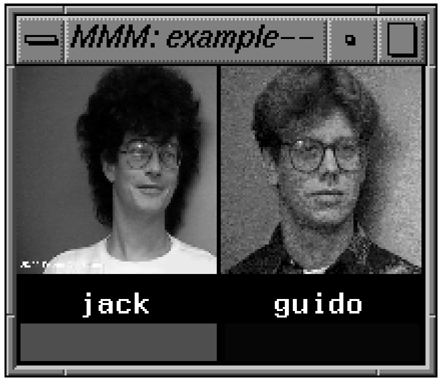
Initially, the user starts a session management tool. This tools allows one to create new sessions and invite other people. Incoming invitations result in a form being presented, and the user having a choice of joining the meeting or not. Whenever you start or join a meeting the correct tools are automatically started.

It is also possible to send invitations for meetings that use a different conferencing tool (like *vat* or *ivs*, for instance). There is also the possibility place a call to a telephone number, using the phone system (audio only, of course:-).

Meeting control

By default, people invited to a meeting get only a single conference window, with the faces and names of all participants and a talk-indicator that lights whenever a person is using the microphone.

The initiator also gets a control panel in which she can select the media types (audio, video, whiteboard) that are needed for this meeting. If they want to, others can also get this panel and change the media used. It is an enforced principle that all participants of a meeting will always get all media streams, so when someone adds a whiteboard everyone in the meeting will get one.



We feel that the policies chosen (exact matching media, media control by all) are suitable for the session types we target at.

Open problems

Because it is impossible to implement everything, some issues are currently not handled by the project:

- Only the audio and (to a lesser extent) video streams are implemented fully. The whiteboard, for instance, currently consists of a simple text-only tool.
- Some work has been done on heterogeneous platforms, but it is currently not a focus point. While a previous version also ran on Sun hardware the current version runs on SGI platforms only.
- Most protocols used are non-standard quick-and-dirty solutions.

Implementation tools

Almost everything is implemented in a language called *Python*, developed locally by Guido van Rossum. Python is an high-level interpreted object-oriented language. In contrast with most high-level languages (like *ABC*, a language on which Python is very loosely based) it has excellent facilities for interfacing to the operating system. Moreover, it is fairly easy to extend the language in an elegant way with modules written in C. This means that sockets, for instance, are first-class python citizens, and that it is fairly easy to write a module to interface to, say, a video grabber interface. An added benefit is that Python more-or-less guides extension module programmers in a direction that will make their modules reusable with very little effort.

The first lesson learned was that the interpretative nature of the language is no real problem in a multi-media environment. Even though there are, of course, some real-time constraints they are very localized and it is easy to write little bits of C code to handle the real-time aspects.

Structure

One of the goals was the eternal *divide et impera* adagium, or the KISS principle. We have distinguished the following tasks:

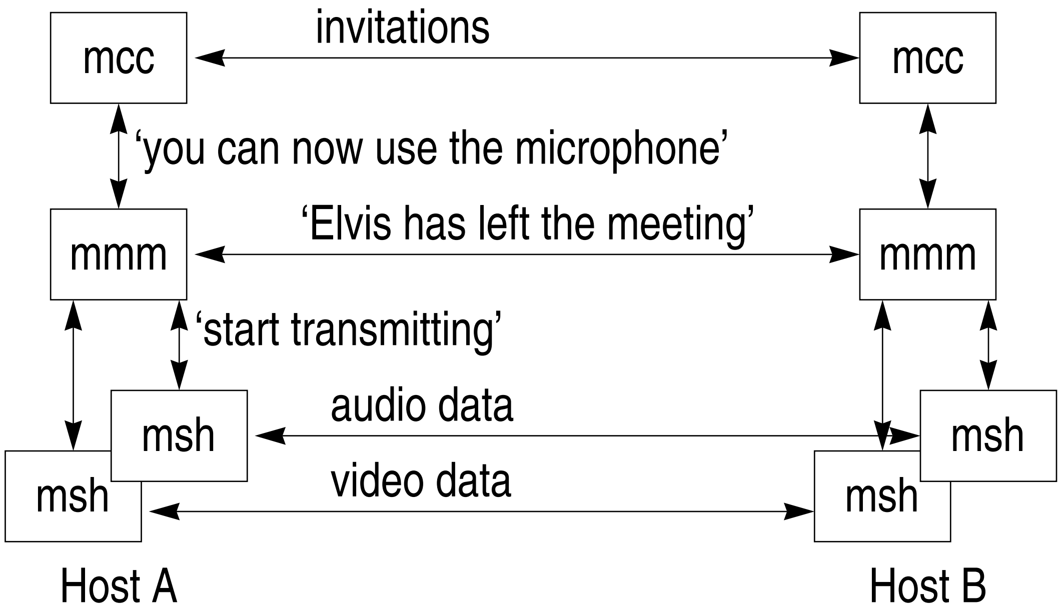
- invitation
- global resource management
  (is there enough network bandwidth for this meeting?)
- local resource management
  (which meeting is using the microphone?)
- per-conference global management
  (who is in the meeting? who is talking?)
- data transport
- data presentation

Since you should never drive a principle too far, we have lumped the top three and the bottom two tasks together. The result is a set of three types of applications:

- *mcc*, the meeting coordinator, which handles invitations and starts conferences
- *mmm*, the conference handler, which supervises a single conference, and
- a number of media stream handlers (*msh*'s for short) that handle a single medium for a single conference.

Interoperation

The communication patterns, with example messages, are as follows:



Because python allows easy marshalling of complicated data structures it is easy to try out various different protocols (since you do not have to write heaps of protocol code every time you change something).

Resource management

Local resource management is only relevent when you are in multiple conferences at the same time. The only local resource we currently manage is the microphone, but the scheme used could also be used to manage the camera or the speaker. When an incarnation of *mmm* detects that the mouse has moved into its window it tells *mcc* that it now has microphone focus. Mcc in turn tells the previous holder of microphone focus that it has lost this. Meetings do not give up the microphone when the mouse moves out of the window, so as soon as the required conference has microphone focus the user can move the mouse anywhere they want (except into the window of another meeting). We do not yet have enough experience with multiple meetings to know whether this simple-minded scheme is good enough or whether we need something better, where meetings can refuse to give up microphone focus, etc.

No work has been done yet on global resource management. Allocating network bandwidth would be a primary candidate to look into, but it is unclear whether a bandwidth allocation scheme would be of any use unless *all* network bandwidth users comply with it.

While not strictly speaking resource management there is another task that *mcc* should handle: inter-conference interactions like merging conferences. It would be nice if you could place a person-to-person phone call during a meeting and then merge the phone call into the meeting.

Per-conference management

The main issue of per-conference management is keeping track of who is in the meeting and what media are in use. We have chosen to implement a single policy: tight centralized control. The main reason for this choice is that much more work is already being done on loose control than on tight control, while tight control is at least as important in our view. The scope of our policy is fairly small numbers of participants like design committee meetings, etc.

One of the policies is that *every* participant should get *everything*, *always*. The rationale behind this is that an electronic whiteboard that can only be seen by half the audience is no good in this scope. It is even worse if half the audience does not see the whiteboard but the other half is not aware of this, so we go to extreme measures to keep the views consistent (as far as killing all whiteboards when one crashes). For continuous media we tolerate packet loss, but we are working on a feedback scheme whereby each user can see how well everyone else hears her (an orange or red indicator overlaid on the persons face in case of moderate or heavy packet loss).

If it should be proven that our policy is too restricting we feel that it is much easier to extend it to allow inconsistent views than the other way around.

Control protocol

We have chosen to use a centralised implementation, based on TCP. All participants have a connection to the originating *mcc* and it implements multicast by simply retransmitting every incoming message over all outgoing connections. The choice for TCP may seem strange, but is ideal for the meetings we are targeting at. With TCP there are no problems of packet loss and the centralised approach forestalls any sequencing problems, etc. Moreover, when one of the participants crashes it is known because the connection will be dropped.

The main disadvantage of this approach is that a crash of the originating *mcc* kills the whole meeting, but in our opinion this can be lived with. The scaling problem of using TCP does not come into play for the meeting sizes we look at.

Session control: Lessons learned

Previous incarnations did not have an invitation mechanism and this was sorely missed. An easy-to-use way to invite people to meetings is an absolute must for a conferencing system. The current invitation scheme is still far from perfect but it is a start. As an example: currently you can accept or decline an invitation, and if you decline there is no easy way to join the meeting later. It would be better if you could decline ‘for now’ and then join later. Coupling the invitations to an agenda system would also be nice.

Older versions of the conferencing tool were monolithic with conference control and media handling integrated into a single program. The split proved to be very handy: it has become much easier to maintain the various parts. The media handlers basically only get told (in a way suitable and specific to them) what destinations their streams should go to and from where they should receive. It is conceivable that you could write a wrapper around other network communication tools (like shared X servers, for instance) that would allow them to function as first-class media stream handlers. This would make the system very easy to extend.

Replacing the UDP based session control by a TCP based version has cleared a few major annoyances, primarily that you now know for sure that everyone you see is also really present. The *everyone-everything-always* dictum strengthens this feeling.

Audio latency

Providing reasonable quality audio has proved to be a major hassle. We strove for a system where users are able to meet without having to use headphones, where you are reasonably free to move your head around and where delays are short enough to be almost imperceptible.

Latency proved to be fairly easy to do. We do as little buffering as possible everywhere, except at one place in the receiver. In other places we simply drop audio when a lag threatens to develop. The receiver buffer is needed to handle jitter in the network. All in all, we get a latency from 0.1 to 0.3 seconds, usually on the low end of the range. This has proved to be acceptable.

We have used fairly large audio packets, 1300 samples. Most network audio tools seem to use much smaller packets but we have not been able to find a reason for that. The bigger packet size reduces overhead (but you must make sure no fragmentation occurs in the network).

All audio processing is done on the sending side, except for mixing audio from multiple sources which is done by the receiver.

Silence suppression and AGC

A large amount of work has gone into silence suppression and automatic gain control. We are now using a scheme where we first debias our incoming audio, then send it through a speech detector and finally through an AGC.

The speech detector looks at signal amplitude only and has two varying thresholds: a silence-to-speech threshold and a speech-to-silence threshold which is 25% lower. There is a feedback loop that lowers the thresholds using a running average during silence and that increases them slightly during speech. The increasing bit will cause an occasional packet to get lost in a talkspurt, but this packet will almost always be during an inter-sentence gap.

The AGC also works on a per-packet basis and tries to keep signal amplitude in a fixed range. It updates its multiplier by at most 10% for every packet, unless the signal threatens to exceed the maximum dynamic range in which case it is immediately lowered substantially. This makes occasional feedback less painful:-)

Echo suppression

Echo suppression is done using a two-sided approach:

- While you are talking the speaker output volume is decreased
- While someone else is talking the speech detector threshold is increased

Previously, we have tried both of these approaches separately but they did not work nearly as well. To forestall feedback with a single approach requires that you completely mute either the microphone or the speaker, and both of these measures can become very frustrating soon. Muting the mike while there is incoming audio means you have no way of interrupting someone holding a long and boring monologue, muting the speaker means that each bit of background noise while you are listening causes you to miss a few words.

The echo suppression can be turned off, so people who use headphones are not bothered by it.

One improvement we still want to do is replace the current abrupt decrease of speaker volume by something more gradual.

Audio: Lessons learned

On the audio front we have tried many solutions for most of the problems sketched before. Silence suppression with fixed threshold, whether the AGC should be before, after or integrated with the speech detector, etc.

A few things need special mention:

- 8Khz is probably not good enough. For men it seems fine, but a female member of our group is considerably more difficult to understand than through 16Khz. We are working on this.
- Some sources said that the number of zero-crossings should also be used in a speech detector. In our case this proved to convey no information whatsoever, probably due to a fairly high level of background noise.
- We have done some work on echo cancellation, by measuring feedback delay and subtracting the attenuated output signal from the incoming audio. This proved to be an utter failure.
- We have experimented with per-speaker background noise when someone is not talking, especially for the benefit of headphone users. So far, we have not succeeded in constructing background noise without obnoxious clicks in it.

An extension that is in the works is to use stereo output and place people at different places in the stereo space. This has been proved by others to be very useful.

Conclusion

There is only one conclusion, really: designing a multimedia conferencing system is difficult and you will get nothing right the first time. Nor the second time. And probably not the third time either.

The corollary of this is that you should prepare yourself to throw things away often, and start from scratch a few times as well. A rapid prototyping language, especially one which facilitates reuse of code, is a tool of immense importance in this area.