XII library from Sun Soft [9] is used for capturing and displaying video frames. Cell-B and MJPEG are the two video formats allowed in our XII version. For audio, the native device driver is employed (/dev/audio). Solaris provides a set of system calls (ioctl) to control this device. Each control allows to set the sampling rate (8 to 44.1 KHz), precision (8 or 16 bits), coding format (linear PCM, u-law and a-law), gain (0 to 255), among other parameters.

The whole application was implemented in approximately 4,000 lines of C++ code, divided as follow: 2,500 lines for the UDP channel, 700 lines for the audio objects (microphone and speaker), 1,000 lines for the video objects (display and camera) and 300 lines for the user-side application. Notice that only 6% was written by the programmer. The code incorporated into the channel and its associated object is platform-independent.

The application objects are linked to a library containing the channel components: factory, stub, binder, protocol adapter and channel controller. Except for the stub, all of these objects are threads within the CORBA server. After the channel has been constructed, the source object (Microphone or Camera) captures a segment of media and simply writes it to the stub. At the sink side, sink object (Speaker or Display) reads from the stub and present this segment of media at the corresponding device driver.

Figure 5 shows the channel object classes in an OMT class diagram [18].

4.1 Scenarios of Utilization

Figure 6 shows an application employing two channels for distributing audio and video from a source to a sink (a point-to-point channel). The client assembles the message at the source side first. Then at each sink node the client builds the sink side and connects to the source. This connection allows the channel controller to know the sinks connected to the source and where they are running. Activating a side of the channel is just a matter of starting the CORBA server housing the channel plus the application.

![Fig. 5: Channel object classes in OMT notation. Boxes are classes (class name plus methods), diamonds state aggregation (part-of) relationships, triangles state generalization (is-a) relationships.](image1)

![Fig. 6: Distribution of audio and video through point-to-point channels.](image2)

Objects. This is achieved by issuing a binding to the channel factory object at the corresponding server (Microphone, Camera, etc.).

In a teleconference application, for instance, point-to-multipoint channels must be employed. A videoconference application builds two source endpoints (one for audio and one for video) and waits the invitation to join from the conference moderator. Suppose the invitation carries the location (node) of all participants. For each participant the invited application builds a video sink (Display) and connects to the participant's video source (Camera). Since video sinks are X Window screens an application can open as many video sinks as the number of sources (see the bottom part of Fig 7).

For receiving audio from many sources the procedure is slightly different. Since there is a single audio sink device per node (the speaker), all the audio channels having sinks at this node must share the application object connected to this device. An application object at the sink side connected to multiple channels to which channel it wishes to receive by selecting the proper stub. The upper part of Fig. 7 illustrates this situation. Building another channel endpoint at a given CORBA server is trivial, being just a matter to ask the factory at that server to instantiate a new channel (stub, binder and protocol adapter objects).

Once the participant finishes the joining procedure it notifies the moderator that informs all members the location of the newmember. At this moment any participant: (i) creates a new video sink and connects it to the newmember's video source; (ii) creates a channel endpoint at the CORBA server hosting the audio sink and connects it to the newmember's audio source. The ingress procedure is now complete.

G.1. Channels allow several videoconference policies to be implemented directly over the communication infrastructure. For example, the conference moderator may allow one participant at a time to speak. In this implementation, this policy is easily realized by the application: except for the speaker, all the audio...