CORBA specifies a large set of distributed services: the CORBA Services and the CORBA Facilities. CORBA Services defines a set of application independent services such as naming, life cycle, security, events and persistence. The CORBA Facilities specifies domain-oriented services such as those for supporting distributed documents and information management.

3 The Channel Architecture

In our implementation each channel endpoint is a CORBA server. This server exports IDL interfaces that allow control operations over the channel. These interfaces are assigned with the application objects, the binder, and the channel controller. The stub and the protocol adapter do not export IDL interfaces (in other words, these objects are manipulated only within the server). In our current implementation we have no interceptors. Figure 4 shows the channel components that are described in the sequel.

3.1 The Channel Factory

The channel factory is responsible for setting-up a channel endpoint. The factory exports a single operation that receives the configuration parameters for the channel, and, based on these parameters, instantiates the remaining channel components. The parameters passed to the factory are: the channel identifier (a name); the transport protocol (UDP, TCP) and parameters (host and port); the source transport address (in case of sinks); and the desired level of quality of service (QoS).

3.2 The Application Object

The application object does not belong to the channel, but resides in the same address space of the channel components for reasons related to efficiency. We implemented the basic multimedia objects: audio, video, speaker, camera, and display.

Typical IDL operations exported by these objects are those related to device control (start, pause, play) and device setup (sampling rate, encoding format, etc.). Application objects are threads that read/write media segments from/to the stub.

In our implementation, application objects format the media using the Real Time Protocol (RTP) [7].

3.3 The Stub

The stub is a passive object instantiated at each side of the channel. It serves as a buffer for data exchanged between the application object and the protocol adapter in a producer-consumer scheme. Worth of mention is the fact that the data stream doesn't cross the binder object, also for reasons related to efficiency.

The stub implements a chain of buffers guarded by semaphores. Application object and protocol adapter access these buffers via two methods: getBufferToWrite, called by the producer and getBufferToRead called by the consumer. These methods enforce a maximum offset in terms of number of buffers between the producer and the consumer. Producer or consumer block if this offset becomes out of range.

The stub exports no interface to the ORB, meaning that its functionalities are available only inside its address space. Moreover, this implementation supposes that all the media processing such as encoding and compression are performed by the multimedia object or by its device driver.

3.4 The Binder

Binders export operations related to channel deactivation, reconfiguration and status reporting. Operations related to flow control (start, stop, resume) are also defined in the binder's interface. These operations are available at this interface because many application objects have no flow control capabilities (start, pause and play operations).

The deactivation operation destroys only the endpoint where the binder is located. Exception to this rule is the destruction of the source endpoint, where all the corresponding sinks are also destroyed. The reconfiguration operations allow a sink endpoint to connect to or disconnect from a source. Status reporting return the latest estimates of delay, jitter, bandwidth and packet error rate at the binder's side.

The binder is implemented as a thread dedicated to the processing of its functions.

3.5 The Protocol Adapter

At the source side, the protocol adapter transfers data read from the stub to its counterparts at the other side(s) of the channel. On the other hand, at the sink side, the protocol adapter reads data from the network and writes into the stub, making such data available to the application object. We implemented an Internet-style protocol adapter that employs UDP to transport RTP packets, transporting media segments.

The protocol adapter is identified by a port number and an Internet address (IP number). It can employ both multicast and unicast addresses, being multicast addresses preferable for point-to-point channels in order to minimize packet duplication in the network.

The protocol adapter is implemented as a thread. The flow control in the channel is performed by creating, suspending and resuming the execution of this thread. Since the channel is a symmetric, the behavior of the protocol adapter depends on the side of the channel (source or sink). Same as the stub, the protocol adapter exports no IDL interface.

3.6 The Channel Controller

The channel controller offers a single point of control for channels. The channel controller keeps a list of all binders in the channel and is employed in a situation where a given operation must take effect in the whole channel (a changing in the encoding format, for instance).

The channel controller is also a thread inside the channel endpoint (server process). This object exports interfaces for setting the references for the binder object in the channel; controlling the media flow; and reporting the status of the overall channel.

4 Implementation Details

The ODP channel was implemented in a network of Sun workstations under the Solaris 2.5 operating system. Orbix-MT1 from Iona Technologies was the chosen CORBA platform [8]. Four classes of application objects were developed: Microphone, Speaker, Camera and Display. These objects define IDL interfaces for device control (start, pause, play), and for device setup (sampling rate, precision, etc.). For video, the

---

1 Multi-threaded Orbix.