Chapter 12

A TINA-based Distributed Environment for Mobile Multimedia Applications

Alexandre S. Pinto, Luis F. Faina and Eleri Cardozo
DCA – FEEC – UNICAMP, PO Box 6101
Campinas, SP, Brazil, CEP 13083-970

Key words: TINA, DPE, CORBA, Mobile Systems, Multimedia Systems

Abstract: This paper describes the design and implementation of a component of the Telecommunications Information Networking Architecture (TINA): the Distributed Processing Environment (DPE). TINA provides concepts, principles, models and standards for the next generation of telecommunication services. These new services take advantage of modern technologies such as high speed networks, multimedia processing, distributed objects, and component-oriented software development. The goal is to devise services that can be introduced, modified and withdrawn as soon as market demands are identified. TINA DPE provides the infrastructure necessary to distribute the service components among service users, providers and retailers. Our DPE implementation offers two basic facilities to the TINA applications: life-cycle and stream facilities. Life-cycle facilities allow distributed objects be deployed and managed transparently, while stream facilities allow application objects to exchange media flows such as audio and video flows.

1. INTRODUCTION

From the last two decades the telecommunication market is experiencing a dramatic growth pushed by demands from large enterprise business to hobbyists at home [1]. However, the current telecommunication infrastructure was designed to carry voice and data, not to provide sophisticated services. The improvement of existing services is currently limited by the lack of computing power available at the switching facilities

J. N. de Souza et al. (eds.), *Managing QoS in Multimedia Networks and Services*

and central office. As a result, the introduction of a new service or modification of an existing one is a slow, costly and labor intensive process.

Back in 1992, with this scenario in mind, about forty companies including telecom operators and equipment suppliers plus computing manufacturers and software suppliers, founded the Telecommunications Information Networking Architecture Consortium – or TINA-C [2]. The chief objective of this consortium is to devise an open architecture (the TINA architecture) for supporting the development and management of telecommunication services, no matter how complex a service would be. In the TINA architecture services are pieces of software with components distributed among users and providers. Being software-based with high degree of platform independence, object-oriented (or more recently, component-oriented), and based on well accepted industry standards, services can be rapidly introduced, adapted and withdrawn according to the market demands.

This work describes the design and implementation of a component of the TINA architecture: the Distributed Processing Environment (DPE). The DPE is an infrastructure responsible for the distribution of the software implementing a telecommunication service. Our DPE implementation is based on CORBA[1] [3] and RM-ODP[2] standards [4]. The highlights of this implementation are the supporting for multimedia flows and object migration.

This paper is organized as follows. Section 2 presents a short introduction to the TINA architecture. Section 3 provides details of the design and implementation of a CORBA-compliant TINA DPE. Section 4 presents how multimedia streams are supported by the DPE as well as how this facility is integrated with the life-cycle facilities. Section 5 presents some implementation issues and an application built above our DPE. Finally, section 6 closes the paper with some concluding remarks and future research directions.

2. OVERVIEW OF TINA

TINA [5] is a software architecture targeted to telecommunication systems and designed according to two principles: distribution and object-orientation. Distribution means that TINA software is decomposed into a set of components, being these components located at different processing nodes. Object-orientation states that the unit of distribution is an object or an aggregation of objects.

---

[1] Common Object Request Broker Architecture from the Object Management Group (OMG).
[2] Reference Model for Open Distributed Processing from ISO.

The TINA architecture is depicted in Fig.1. Each processing node is based on some hardware architecture that runs a Native Computing and Communication Environment, the NCCE. NCCE can be viewed as a dedicated or general-purpose operating system plus a stack of network protocols. Each processing node has an implementation of the DPE running on it.

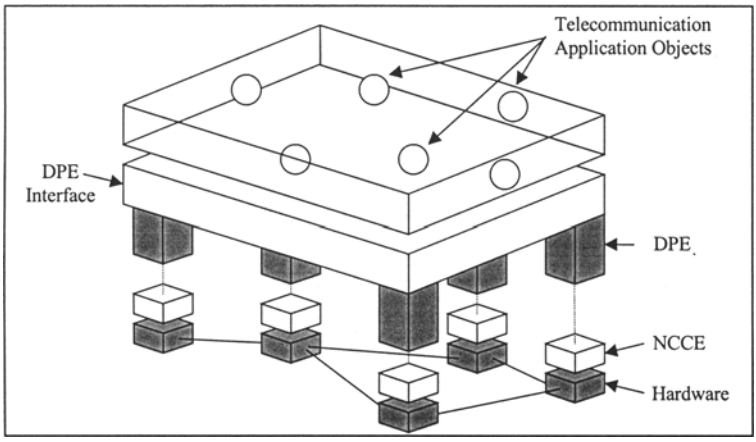

Figure 1. TINA Environment [5]

The DPE offers a homogeneous interface to the TINA applications (Fig.1). Some DPE functionalities may be missed at some NCCE. For instance, a NCCE dedicated solely to management tasks can be deprived from multimedia stream facilities.

The TINA DPE [6] has two major functions: distribution (deployment) and communication. The distribution function defines the unit of distribution as *engineering computational object* - eCO. An eCO has one or more interfaces. eCOs are instantiated from *templates*, something similar to classes in object-oriented programming languages. A template states rules for object instantiation such as the initial state and interfaces. For management purposes, objects are aggregated into *clusters*. A cluster is a unit of spatial location, activation/deactivation and migration. eCOs can be created and removed from a cluster during the cluster's lifetime. Clusters live inside *capsules*, the TINA's unit of resource allocation. An interesting (and rarely implemented) property of clusters is migration: a cluster can migrate from a capsule to another carrying all of its objects. The last component in the deployment model is the *node*, a unit of network connectivity and network management. A node provides DPE support and manages its computing resources autonomously.

In order to interact, objects must bind their interfaces. The DPE communication function defines two types of interfaces: operational and stream interfaces. The interaction that occurs at an operational interface is structured in terms of invocations of one or more operations, and possibly responses to these invocations. In a stream interaction, the information exchange occurs in the form of unidirectional media flow (e.g., audio and video flows).

In order to support interactions between objects located at different clusters, a channel is defined (Fig.2). A channel is an infrastructure providing mechanisms for supporting distribution transparency (access, location, etc.) plus an interface for management and control purposes (Fig.2).

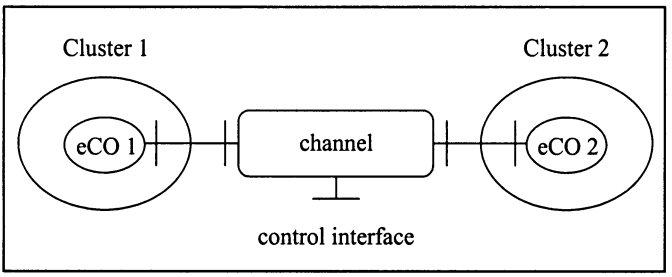

Figure 2. A channel binding two eCOs

3. DPE ARCHITECTURE

The architecture of the implemented DPE is shown in Fig.3. The DPE is built over a CORBA platform [7]. A CORBA platform allows interactions between objects in a distributed heterogeneous system. DPE instances are connected through the OMG's Internet Inter-ORB Protocol (IIOP), a standard for ORB interworking [3].

The two main facilities available at our DPE are the stream [8] and the life-cycle [9] facilities. Stream facilities allow the telecommunication services to manipulate multimedia streams (flows), while life-cycle facilities allow the service components to be distributed and managed across the network. Stream facilities are fully compliant with the OMG's *Control and Management of Audio/Video Streams* [10] and the life cycle facilities were defined according the TINA DPE specifications [5] [6].

Back to Fig.3, the life-cycle facilities provide functionalities for the management of eCOs, clusters and capsules (including cluster migration), and the stream facilities provide functionalities for the establishment and

management of channels. All the DPE functionalities, available at the DPE interface, are accessed through an object request broker (ORB).

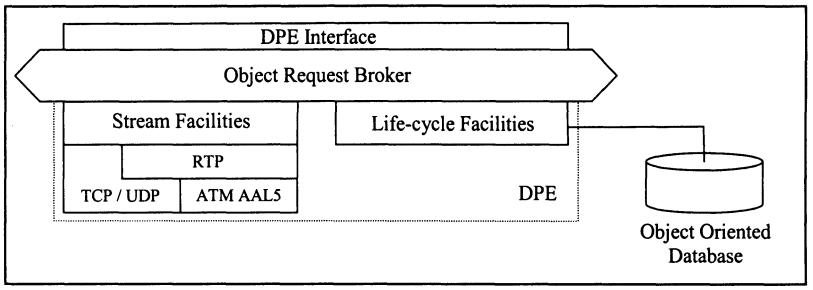

Figure 3. DPE Architecture

3.1 Support for Life-Cycle

The support for life-cycle in our TINA DPE is realized through management interfaces at object, cluster, capsule and node levels.

Each interface defined in a TINA object is realized through a CORBA object. A special CORBA object, the eCO manager, defines an interface for management purposes (the remaining interfaces are referred as application interfaces). This management interface has the following IDL description:

```
interface eCOManager {
  long addInterface(in Object interface_ref);
  long removeInterface(in Object interface_ref);
  sequence<Object> getInterfaces();
  long checkpoint(in string template);
  long recover(in string template);
  long Delete();
  long deactivate(in string template);  };
```

The operation `checkpoint()` saves the state of the CORBA object in a template; `recover()` restores the state of the CORBA object from a template; `Delete()` acts as a destructor for the CORBA object, releasing all the resources allocated to the object; and `deactivate()` is equivalent to a `checkpoint()` followed by a `Delete()`.

Typically, templates point to database entries that store the state of objects in persistent memory. The `eCOManager` interface performs

management functions within a TINA object. This interface is application independent, being its implementation provided by the DPE.

Application interfaces are added to and removed from an eCO via `addInterface()` and `deleteInterface()` operations, respectively. These operations receive as input parameter the interface reference (that is, the CORBA object reference servicing the interface). `getInterfaces()` returns a sequence of all application interfaces currently attached to the TINA object. The remaining operations simply call their counterparts at each CORBA object servicing the application interfaces.

The application is responsible also to provide a factory for object instantiation from templates.

The management of clusters is provided by cluster managers implementing the following IDL interface:

```
interface ClusterManager {
  eCOManager makeObject(in string object_name);
  sequence<eCOManager> getObjects();
  long checkpoint(in string template);
  long recover(in string template);
  long Delete();
  long deactivate(in string template);  };
```

The operation `makeObject()` installs a new object in a cluster, returning a reference of its eCO manager. The input parameter is a name from which the object is identified in the cluster. This new object is created without any application interfaces. `getObjects()` returns the references of the object managers in the cluster. The remaining operations, `checkpoint()`, `recover()`, `deactivate()` and `Delete()` call their corresponding operations at each object manager in the cluster.

The capsule management interface is given below:

```
interface CapsuleManager {
  ClusterManager makeCluster(in string cluster_name);
  sequence<ClusterManager> getClusters();
  long reactivate(in string cluster_name, in string template);
  long migrate(in string cluster_name, in string capsule_name,
in string node_name);  };
```

The operation `makeCluster()` creates a new empty cluster within the capsule, returning a reference of its cluster manager. The input parameter is

a name from which the cluster is identified in the capsule. `getClusters()` returns the references of the cluster managers in the capsule. `reactivate()` installs a new cluster from a template generated during a cluster checkpoint. Figure 4 illustrates a migration scenario accomplished through deactivation followed by reactivation of the migrating cluster.

Node management is performed by node managers implementing the following IDL interface:

```
interface NodeManager {
  sequence<CapsuleManager> getCapsules();  };
```

This interface defines only one life-cycle operation, `getCapsules()` that returns all the capsules created within the node.

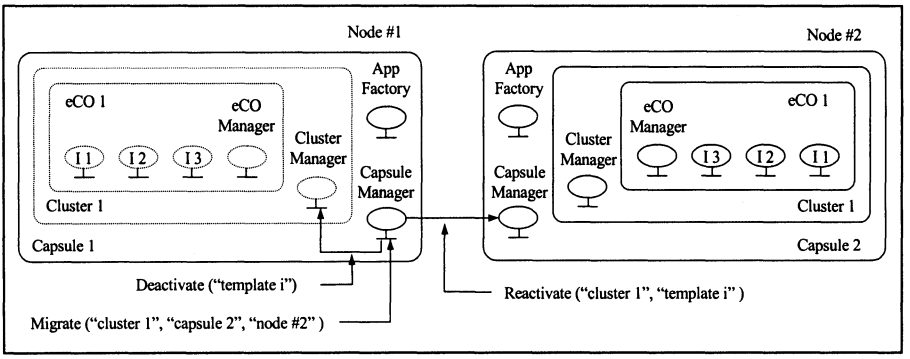

Figure 4. Cluster migration.

3.2 Support for Object Binding

Implicit binding is provided by the object request broker (ORB), needing no additional support from the DPE. The support for explicit binding through channels is an important component of the DPE. As stated before, in our implementation, channels are realized above an infrastructure based on the OMG standard *Control and Management of Audio/Video Streams* (AVStreams) [10]. This infrastructure allows the establishment and management of media flows, and its functionalities are related to the channel control interface shown in Fig.2. AVStreams, as detailed in the next section, is fully integrated with the life-cycle facilities. This means, for instance, that when an eCO is destroyed, the channel endpoints connected to it are also destroyed. Also, when an eCO migrates with its cluster, the channel endpoints are reconstructed in the cluster's new location.

4. DPE STREAM FACILITIES

Streams are facilities for supporting continuous media transfer. A stream aggregates a set of *flow connections* and terminates on a *stream endpoint*. Each flow connection is unidirectional and ends on a *flow endpoint* (FEP) within a stream endpoint. Flow endpoints can be a flow consumer or flow producer.

Multimedia devices abstract physical or logical devices that generate or consume media segments. Multimedia devices are bounded through streams in order to establish a media flow between a media producer and one or more media consumers. The interface `MMDevice` defines operations related to multimedia devices.

Inside multimedia devices two objects are responsible for controlling the media transferring: the stream endpoint object and the virtual device object. Stream endpoints provide operations related to network transport (connections, quality of service, etc.), while virtual devices provide operations related to device configuration (media formats, sampling rates, etc.). The interfaces `StreamEndPoint` and `VDev` abstract stream endpoints and virtual devices, respectively.

Streams are controlled by a control object that provides operations for flow control (`start()`, `stop()`, `destroy()`) and multimedia device bindings. The interface `StreamCtrl` implements such operations.

Figure 5 shows a unidirectional audio stream connecting a microphone to a speaker. The `StreamCtrl` interface, pictured in Fig.5, is equivalent to the channel control interface shown in Fig.2.

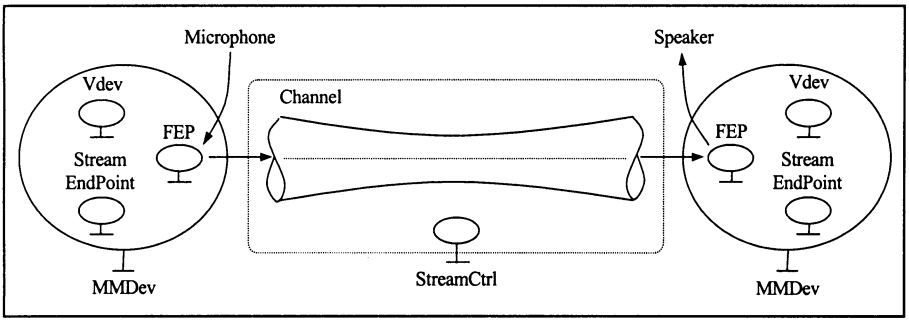

Figure 5. A basic audio stream configuration

As stated before, our DPE implementation provides mobility of clusters through deactivation and subsequent reactivation on a different capsule. One important feature of our DPE is the incorporation of mobility to channels. When a cluster migrates, all channel endpoints are equally deactivated and

reactivated at the new cluster destination. This feature allows a TINA session to be suspended at one site and resumed on a different site.

In order to incorporate mobility to channels, we specialized our AVStreams implementation, mainly the stream controller (`StreamCtrl`) and the multimedia device (`MMDevice`) objects. The idea is to incorporate mobility functions to stream controllers and checkpointing/recovering functions to multimedia devices. Such functions are absent in the current OMG AVStreams specification. The extended `MMDevice` IDL interface is given below:

```
interface AppMMDevice : AVStreams::MMDevice {
  long checkpoint(in string template);
  long recover(in string template);
  long Delete();  };
```

This extended interface just turns an AVStreams multimedia device object into an engineering computational object (eCO) by incorporating the `checkpoint()`, `recover()` and `Delete()` operations.

The extended `StreamCtrl` IDL interface is presented below:

```
interface ChannelCtrl : AVStreams::StreamCtrl {
 long configureChannel (in AVStreams::streamQos the_qos,
      in AVStreams::flowSpec the_spec);
 long configureEndPoints (in eCOManager a_eco, in AppMMDevice
      a_party, in eCOManager b_eco, in AppMMDevice b_party);
 long changeEndPoint(in eCOManager eco, in AppMMDevice party);
 long deactivateChannel();
 long reactivateChannel();  };
```

This extended interface allows the configuration (`configureChannel()`) of the channel by stating the quality of service parameters (e.g., network bandwidth and maximum delay) and the specification of the flows (e.g., PCM audio and MPEG video). `configureEndpoints()` establishes the binding between two multimedia devices, assigning them to their respective eCOs. `changeEndPoint()` allows the re-assignment of a multimedia device to a new eCO. This operation is performed when an eCO attached to a channel is reactivated after migration. `deactivateChannel()` and `reactivateChannel()` operations are performed during the migration process.

Two more operations are related to channel creation and destruction and are part of the node interface (as specified by RM-ODP):

```
interface NodeManager {
  long createChannel
    (in eCOManager a_eco, in AppMMDevice a_party,
     in eCOManager b_eco, in AppMMDevice b_party,
     in AVStreams::streamQos the_qos,
     in AVStreams::flowSpec the_spec,in ChannelCtrl the_ctrl);
  long destroyChannel(in ChannelCtrl the_ctrl);  };
```

TINA applications call `createChannel()` in order to establish a stream binding between two eCOs. The application passes the references of the multimedia devices (a_party and b_party) and the eCO managers assigned to the eCOs to be bound. QoS parameters and flow types (the_qos and the_spec) may also be specified (null parameters means "no QoS specified" and "all flows", respectively). The last parameter is a reference of a channel controller that will manage the channel. Each channel type supported must implement a channel controller of that type. Our DPE has two channel controllers implemented: one for PCM audio and one for CellB video. `createChannel()` returns a flag indicating success or failure. Upon success, the application uses the channel control object to start, pause and destroy the channel.

When a cluster migrates, the DPE takes the following actions with respect to channels. Each eCO manager maintains a list of channel controllers managing the channels connected to the object. When the object is deactivated, the eCO manager calls the `deactivate()` operation on the channel controller in order to de-allocate the resources assigned to the channel. When the object is reactivated in the destination capsule, the eCO manager re-establishes the channel by calling `changeEndPoint()` and `reactivateChannel()` on the channel controller. Notice that the channel controller is not deactivated during migration of channel endpoints.

In case of point-to-point channels both channel endpoints are destroyed and reconstructed during migration. For point-to-multipoint channels, only the migrating endpoint is destroyed and reconstructed during migration.

5. IMPLEMENTATION ISSUES

The TINA DPE reported in this paper was implemented on the following computing infrastructure. Processing nodes consist of a set of Sun Sparc 5 and Ultra workstations with OC-3 ATM network interface card (155 Mbits/s) connected through an ATM switch from Xylan Corporation. The Solaris 2.6 operating system from Sun Soft runs on each processor. Orbix 2.3 [11], a CORBA 2.0-compliant platform from Iona Technologies, and ObjectStore 5.0 [12], a distributed object-oriented database from Object

Design, are the major software products employed in this project. Orbix provides the major CORBA components (IDL compiler, ORB, implementation and interface repositories and locator), while ObjectStore provides persistence functions necessary for checkpointing and recovering of objects, clusters and channels.

At each node runs a CORBA server (servant) implementing the `NodeManager` interface. Capsules are Solaris processes with a servant implementing the `CapsuleManager` interface. This servant executes on an exclusive thread. Cluster and eCO managers are also servants running on exclusive threads. Application servants are distributed through any number of threads. All of these threads execute within the capsule's address space. This implementation follows exactly the RM-ODP where capsules provide the resources for clusters and objects.

In order to illustrate the DPE support for migration, let us consider a "desktop telephone" service[3] (Fig.6). Before the establishment of the audio stream supporting the conversation, two capsules are created on each participating node. The application at the caller side calls `createChannel()` on its node interface in order to establish an audio channel with two flows connecting the multimedia devices (a pair of microphone and speaker at each side). These devices derive from the interface `AppMMDevice`. An object that implements the interface `audioChannelCtrl`, a specialization of the interface `ChannelCtrl` for audio streams, controls the audio channel.

Suppose that one party decides to move to a different node. The application can then build an empty capsule at this node and call `migrate()` on the actual capsule passing the new capsule as target. After the cluster migration completes, the channel is re-established and the application resumes. It is important to notice that TINA applications hide the DPE functions from the final users.

6. CONCLUDING REMARKS

This paper presented a CORBA-based implementation of a major component of the TINA architecture, the Distributed Processing Environment (DPE). The implementation has the following highlights: support to deployment and management of nodes, capsules, clusters and objects; support to stream binding, allowing application objects to exchange media flows; support to migration, allowing clusters to migrate from one capsule to another; multi-threaded implementation, allowing different levels of parallelism among the objects.

---

[3] The telephone set is a desktop computer and the telephone networks is a data network.

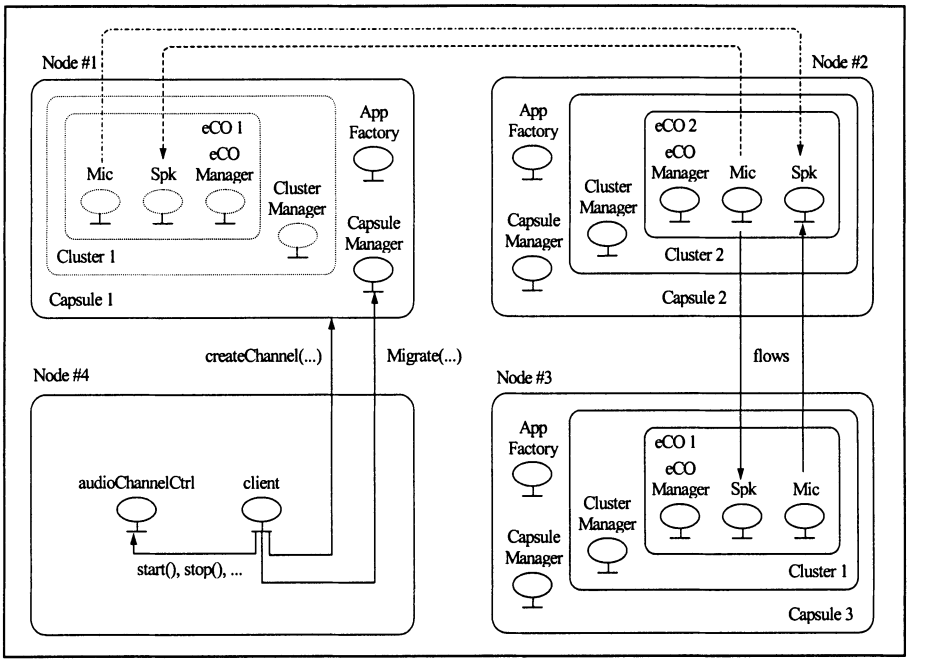

Figure 6. Desktop telephone application.

Among the applications of interest, the offering of telecommunication services over the Internet is receiving special attention. The increasing of speed in both on access networks and backbones is making the Internet the most promising infrastructure for new telecommunication services. Since the TCP/IP protocol stack provides only connectivity services, the integration of TINA and Internet can turn the Internet in a truly multi-service global network. Another point of interest is to employ some CORBA 3 [13] features, mainly, objects supporting multiple interfaces[4] and quality of service control. These features are not available in the current CORBA development platforms.

ACKNOWLEDGMENTS

This research is being supported by the following Brazilian funding agencies: CAPES which provides MSc and PhD scholarships to the first and second author, CNPq (grant 300723/93-8) and FINEP (grant 1588/96).

[4] Multiple interfaces allows a direct mapping between TINA and CORBA objects.

REFERENCES

[1] R.W. Lucky. "New Communications Services – What Does Society Want?", Proceedings of the IEEE, 85(10), October 1997.

[2] Fabrice Dupuy, Gunnar Nilson, and Yuji Inoue. "TINA Consortium: Toward Networking Telecommunications Information Services", IEEE Communications Magazine, 33(11): 78-83, November 1995.

[3] Object Management Group. "CORBA/IIOP 2.3.1 Specification", Technical Report formal/99-10-07, 1999. URL at http://www.omg.org.

[4] ISO/IEC 10746 / ITU-T Draft Rec. X.901, X.902 & X903. "ODP Reference Model Part 1, Part 2 & Part 3", Technical Report, June 1995.

[5] H. Berndt et al. "The TINA Book". Prentice Hall Europe, 1999.

[6] TINA-C. "Engineering Modelling Concepts - DPE Architecture", Technical Report, TINA Consortium, December 1994. URL at http://www.tinac.com/.

[7] S.Vinoski. "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments", IEEE Communications Magazine, 14, February 1997.

[8] L.F.Faina, E.J. Oliveira, R.C.M. Prado and E. Cardozo. "Developing Multimedia Applications with the OMG Streaming Framework". In Proc. of the ICC'99 - MAST'99 (Mini Conference on Multimedia Applications, Services and Technologies), Vancouver, Canada, June 1999. URL at ftp://ftp.dca.fee.unicamp.br/pub/docs/eleri/Mast99.pdf

[9] A.S. Pinto, E.J. Oliveira, L.F. Faina and E.Cardozo. "TINA-based Environment for Mobile Multimedia Services". In Proc. of the TINA'99, Hawaii, USA, April 1999.

[10] Object Management Group "CORBAtelecoms: Telecommunications Domain Specification", Technical Report formal/98-07-12, OMG, October 1997. URL at http://www.omg.org/.

[11] IONATechnologies Ltd., "Orbix 2.3c MT - Programming Guide", November 1997.

[12] Object Design. "ObjectStore - C++ API User Guide - Release 4", June 1995.

[13] Jon Siegel, "CORBA 3 Fundamentals and Programming", John Wiley & Sons, second edition, 2000.