The Tangram DPE - A Distributed Processing Environment in a Heterogeneous Corba 2 World

M. Khayrat Durmosch, Klaus-D. Engel
GMD FOKUS, Hardenbergplatz 2, D 10623 Berlin, Germany
email: {durmosch,engel}@fokus.gmd.de

Abstract

Within this paper we will describe the usage of a Distributed Processing Environment based on CORBA 2 with respect to distribution transparencies. We will sketch a multimedia telecommunication service (T-MMCS) based on TINA-C concepts and our experience using CORBA 2 implementations (Orbix and HPDST) as DPE for this application.

At the moment the CORBA Scene is changing very rapidly. New products are coming up and interoperability between products is increasing due to corrections in the implementations. More and more CORBA Services are supported. The CORBA Standards are also extended and more services are defined. This changes will also happen in the time between submitting this paper and the time it will be presented to the conference audience. Therefore we will present the newest developments that took place and our newest experiences on the conference.

1. Introduction to the TANGRAM project

The Tangram project began in 1995 and is a GMD FOKUS project in the R&D program of the DeTeBerkom, a subsidiary of the Deutsche Telekom. Its goal is to validate the TINA-C recommendations for the architecture of services, computing and management. The project will build on heterogeneous Distributed Processing Environment (DPE) for the support of some multimedia telecommunication services including, initially a Multimedia Conferencing application (T-MMCS).

1.1 TINA-C

Due to the influence of the emerging distributed computing technology the telecommunication industry is about to take a new approach for the development, construction and operation of software for telecommunication systems. A consortium of about forty international companies (service providers, network operators and telecommunication vendors) founded in1993 the Telecommunication Information Networking Architecture Consortium (TINA-C). TINA-C‘s goal is the development of on an environment comprising open distributed processing platforms positioned in telecommunication systems which include a variety of existing and forthcoming network technologies [7]. Much of this work is based upon the Reference Model of Open Distributed Processing (RM-ODP) [6]. On the other hand the OMG CORBA activities [8] [9] have also greatly influenced the TINA-C distributed processing environment and its services.

1.2 T-MMCS

Within the Tangram project we are developing a distributed telecommunication application for a multimedia communication service (T-MMCS), which adheres to suggestions made in the TINA-C Service Architecture [10].

The T-MMCS enables the exchange of audio and video data between a group of participants. During a session new participants can be invited to join or active participants can leave. The application is designed to fit commercial situations, where a customer books for his users the ability to use a service by making a contract with a service provider. For example, a company may subscribe with a service provider to provide a telephony service for its employees. This contract determines customization and also the accounting, tariffing and billing when the service is used.

Within Tangram the T-MMCS is implemented to provide a near realistic scenario for the project specific DPE, and is used as a starting point for the development of an abstract object-oriented framework which supports a service design architecture.

A complete description of the design and the components of the T-MMCS is beyond the scope of this paper. We can only briefly discuss the type of service that would

Proceedings of The Thirtieth Annual Hawwaii International Conference on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE

use the Tangram DPE. For more information on the design and implementation of the T-MMCS, see companion paper [14].

The T-MMCS was developed using ODP Viewpoints.

The enterprise viewpoint results in an access session, during which a service provider is contacted and usage proformas are checked; a service session, where users can be invited and an agreement is established between the parties on the mutual technical environment; and communication sessions during which the audio and video data is mediated through the network.

Computationally the results identify components and their relations with clearly distinguishable responsibilities with respect to the entire T-MMCS (Figure 1)

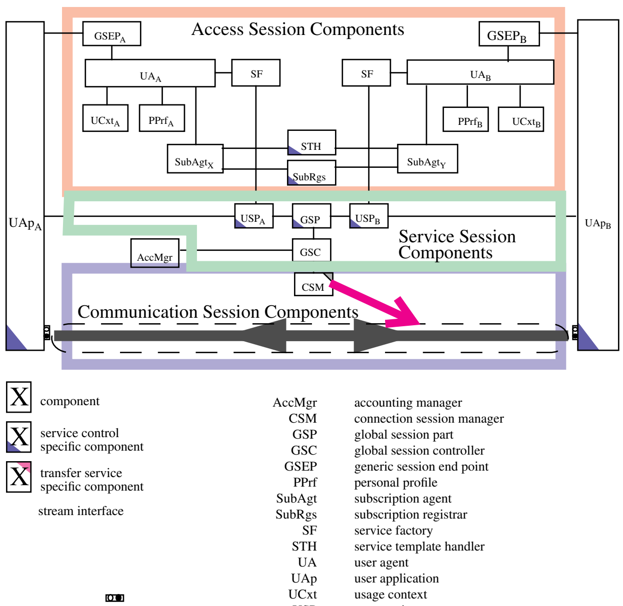

Fig. 1: Component configuration of T-MMCS

Because some phases, like the access session, appear to be needed in other services, a wide reusability can be expected for those components which are not service specific.

The details of the components and their relationships and interactions is beyond the scope of this paper. The Tangram DPE will be used to support the components and their interrelations. The components and their interfaces will be specified using the TINA-C Object Description Language (ODL).

1.3 The TANGRAM DPE

The Tangram DPE should be used to support the T-MMCS on a variety of hardware platforms including SUNs, Windows PCs, Apple Macintosh. The platform products actually used are the CORBA implementations Orbix of IONA [12] and the HP-Distributed Smalltalk of Hewlett-Packard [13]. The Tangram DPE should form a distributed processing platform that offers distribution transparencies to the components which use it and masks the heterogeneity of the underlying systems (Figure 2).

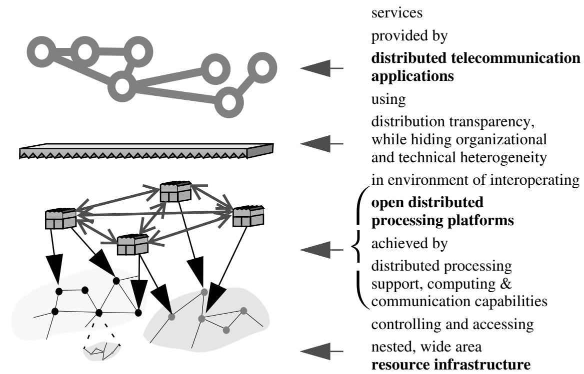

Fig. 2: Scenario of open distributed processing platforms and distributed telecommunication applications.

Transparencies that should be supported by DPE Functions are sketched below. Basic CORBA communication and the CORBA Services are a good starting point to support this transparencies.

Access transparency, masks differences in data representation and invocation mechanisms between heterogeneous computer architectures, programming models, and networking protocols to enable interworking between objects. Access transparency is supported by the basic CORBA communication mechanisms.

Location transparency, masks the use of information about location in space when identifying and binding to interfaces. This transparency provides a logical view of naming, independent of the actual physical location. Location transparency is supported by the CORBA Naming Service.

Failure transparency, masks from an object the failure and possible recovery of other objects (or itself) to enable fault tolerance. When this transparency is provided, the designer can work in an idealised world in which the corresponding class of failures does not occur. There is no CORBA Service at the moment which directly supports this transparency.

Migration transparency, masks from an object the ability of a system to change the location of that object. Migration is often used to achieve load balancing and to

Proceedings of The Thirtieth Annual Hawaii International Conference on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE

reduce latency. The CORBA Lifecycle Service can support this transparency.

Relocation transparency, masks the relocation of an interface from other interfaces which are bound to it. Relocation allows system operation to continue even when migration or replacement of some objects creates temporary inconsistencies in the view seen by their users. The CORBA Lifecycle Service supports this transparency.

Replication transparency, masks replication of components, thereby enhancing performance and availability. There is no special CORBA Service to support this transparency in the moment.

Concurrency transparency, masks the use of a group of mutually compatible objects to support an interface. Concurrency transparency hides the effect of the parallel execution of applications. CORBA Concurrency service can be used to support this transparency.

Persistence transparency, masks from an object the deactivation and reactivation of other objects (or itself). Deactivation and reactivation are often used to maintain the persistence of an object when the system is unable to provide it with processing, storage and communication functions continuously. CORBA Persistency Service could support this transparency.

Transaction transparency, masks the coordination of activities among a configuration of objects to achieve consistency. CORBA Transaction Service supports this transparency.

The complete Specification of the CORBA communication facilities and the CORBA services can be found in [8] and [9].

2. CORBA 2 Interoperability - Theory

CORBA 2 Interoperability part adds multivendor interoperability to the ORBs. This includes initialization services, and enhancements to the Interface Repository. With CORBA 2, an ORB can broker interactions between objects that reside within a single process as well as between objects that globally interact across multivendor ORBs and operating systems.

2.1 OMG CORBA 2 Interoperability Compliance

An ORB is considered to be interoperability-compliant when it meets the following requirements, see Fig. 3:

- It supports the standard ORB APIs defined by CORBA2/Core: the Dynamic Invocation Interface, the Dynamic Skeleton Interface, the specification around the Object References converting and manipulating operations, and the ORB and OA Initialization and Initial References specification [CORBA2: p9-5]
- A native or half-bridge support of the Internet Inter-ORB Protocol (IIOP) [CORBA2 Chap 12], which specifies how General Inter ORB Protocol (GIOP) messages are exchanged over a TCP/IP network.

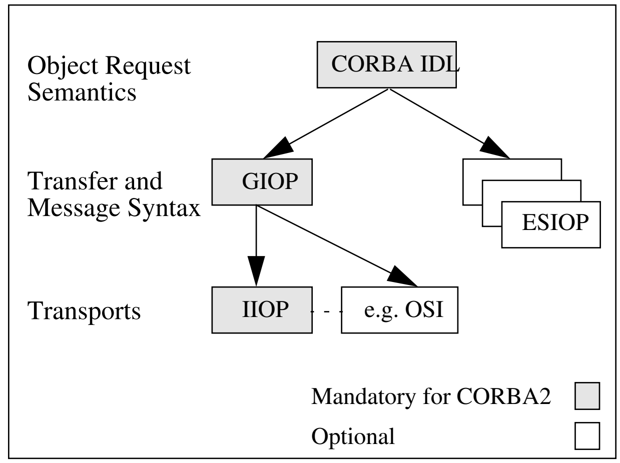

Fig. 3: CORBA 2 Inner-ORB Protocol Relationships

Additional support for several Environment-Specific Inter-ORB Protocols (ESIOPs) is optional. In the future the GIOP may also be mapped to different transports. The IIOP defines a transfer syntax and message formats.

2.2 Core CORBA and Interoperability extensions

CORBA2/Core consists of the extended *CORBA 1.2* specification [8] without the C language binding, the extensions to support the CORBA Services 2 [9], the Interface Repository, the Initialization specification, and the Inter-ORB Bridge support. Special interoperability extensions are the GIOP with IIOP, and the ESIOP DCE IOP. This paper focuses on the components mandatory for CORBA2 interoperability.

2.2.1 The Interface Repository

The IR as a logical part of the ORB provides persistent storage of interface definitions. It manages and provides access to a collection of interface (object) definitions specified in IDL. These object definitions are available to an ORB as “interface objects” accessed through IDL-specified interfaces. In particular the object definitions in the IR assist the ORB in providing interoperability between different organizational domains (e.g. different types of systems), and between different technological domains (e.g. different ORB implementations).

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE

The information within the IR is used for connecting ORBs together. A key concept for supporting this is to assign a unique *RepositoryId* to an object interface. The same RepositoryId in different repositories refers to the same object interface type. When an object is passed in a request from one ORB to another, the receiving ORB may get the repository id from the repository of the sending ORB. The receiving ORB may use this id to look up the interface in its repository.

The RepositoryId concept requires a mechanism to include RepositoryIds with published IDL specifications. A convention is specified for using `#pragma` directives to annotate IDL specifications with these Ids.

2.2.2 The Initialization Specification

This specification describes the APIs for obtaining ORB and object adapter (OA) pseudo-objects[1], and other initial objects available to a CORBA application for bootstrap purposes.

Every application needs a mechanism for initialisation in a CORBA environment and to get ORB and OA pseudo-object references for use in future ORB and OA operation calls. The initialisation operations must be in a specific order: ORB initialisation operations are called before those of the OA.

ORB initialisation

The operation to initialise an application in the ORB and get to its pseudo-object reference is not performed on an object. This is because applications do not initially have an object on which to invoke such operations. This operation is an application's bootstrap call into the CORBA world:

```
//PIDL
module CORBA {
  typedef string ORBid;
  typedef sequence <string> arg_list;
  ORB ORB_init (inout arg_list argv,
                    in ORBid orb_identifier);
};
```

Once an ORB initialisation has been completed, its pseudo-object reference is returned and can be used to obtain OA references for that ORB.

Object Adapter Initialisation

An ORB may have zero or more object adapters (OA) associated with it. Servers must have a reference to an OA pseudo-object to access its functionality. Given an ORB reference, an application must be able to initialise itself in an OA environment and get the pseudo-object reference of the OA from the ORB.

OA do not necessarily share a common interface, although they have to provide the functionality of the BOA. Therefore it is necessary to provide an initialisation function for each OA type separately. This leads to a template suggestion for OA initialisation, and the generated BOA initialisation operation from that template:

```
//PIDL
module CORBA {
  interface ORB {
    typedef sequence <string> arg_list;
    typedef stringOAid;

    // Template for OA initialisation operations:
    // <OA> <OA>_init (inout arg_list argv,
    //  in OAidoa_identifier);

    BOA BOA_init (inout arg_list argv,
       in OAid boa_identifier);
  };
};
```

The allocation of `OAids` is in the responsibility of ORB administrators and is not managed by the OMG. Once the `BOA_init` operation has returned the BOA, it is assumed to be initialised for the application object.

Initial Objects

The references of the initial object services have to be obtained by applications in a portable way. This is done by a simplified local version of the Naming Service which applications can use to obtain a small well defined set of object references of initial objects:

```
//PIDL
module CORBA {
  interface ORB {
    typedef string ObjectId;
    typedef sequence <ObjectId> ObjectIdList;
    exception InvalidName {};

    ObjectIdList list_initial_services ();
    Object resolve_initial_reference
              (in ObjectId identifier)
              raises (InvalidName);
  };
};
```

`ObjectIds` are strings which identify the object whose reference is required. The OMG defines which initial services are required by applications, and specifies the names for those services. Currently two ObjectIds are defined: InterfaceRepository and NameService.

1. Pseudo-objects have to be understood as objects defined in IDL only to ensure the well defined language mapping. They are not normally distributed.

2.2.3 ORB Interoperability Extension

Interoperability between several heterogeneous ORBs means a transparent crossing of domain boundaries. Conceptually this is done by translating the necessary information when an object request traverses domain boundaries. This translation is provided by a mapping or *bridging* mechanism which resides at the boundary between the domains, and which transforms requests expressed in terms of one domain's model into the model of the destination domain. The object models of the two domains need to be compatible to achieve full interoperability. This should be the case if both domains comply with the CORBA Object Model.

Interoperable Object References - IORs

Clients should be able to use object references to invoke operations on objects in other ORBs. The client does not need to distinguish between references in local or remote ORBs. An *Interoperable Object Reference - IOR* has been defined to provide this transparency. This new data structure should only be used when crossing object reference domain boundaries within bridges, and should not be visible to application level ORB programmers.

The IORs have one tagged profile for each protocol is supports. These profiles encapsulate all the basic information that a protocol needs to identify an object. Each profile has an unique numeric tag, assigned by OMG. The tags for the IIOP and the use of *multi component protocol profiles* have already been defined. The IDL specification of the IOR is shown below:

```
module IOR {
  //
  // Standard Protocol Profile tag value
  //
  typedef unsigned longProfileId;
  const ProfileId TAG_INTERNET_IOP = 0;
  const ProfileId TAG_MULTIPLE_COMPONENTS = 1;
  struct TaggedProfile {
    ProfileId tag;
    sequence <octet>profile_data;
  };
  //
  // an IOR is a sequence of object-specific
  // protocol profiles, plus a type ID.
  //
  struct IOR {
    string type_id;
    sequence <TaggedProfile> profiles;
  };
      ...
};
```

The `TAG_MULTIPLE_COMPONENTS` tag indicates that the value encapsulated in `profile_data` consists of a list of protocol components, which indicate ORB services which are accessible using that protocol. Section Interoperable Object References - IORs describes the composition of the `profile_data` structure when IIOP is used.

IORs are created from conventional object references when required to cross a referencing domain boundary. ORBs implement object references they use internally in what ever form they find appropriate. Bridges will normally use IORs to mediate transfers. The IORs could be stringfied or externalized in a parsable string form and vice versa. A stringfied IOR representation is specified for that purpose.

General Inter-ORB Protocol GIOP

The GIOP specification consists of the Common Data Representation (CDR), GIOP Message Formates, and GIOP Transport Assumptions.

The CDR is the transfer syntax into which the OMG-IDL data types are mapped for transfer between Inter-ORB Bridges. The IDL datatypes are represented in an octet stream, which is an arbitrary finite long sequence of eight-bit values with a well defined beginning. GIOP defines two distinct kinds of octet streams: messages and encapsulations. Messages are the basic units of information exchange, encapsulations are octet streams into which IDL data structures may be marshalled independently from any message context.

GIOP defines a special message format for facilitating object requests, locating object implementations and handling communication channels. The GIOP also defines a mechanism for identifying and passing service-specific context information as „hidden parameters“. Seven message types and potential originators (client or server) have been defined:

Table 1: GIOP Messages and originators

Message Type	Originator	Value
Request	Client	0
Reply	Server	1
CancelRequest	Client	2
LocateRequest	Client	3
LocateReply	Server	4
CloseConnection	Server	5
MessageError	Both	6

All messages begin with a IDL specified header which clearly identifies the message type, byte order, the

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE
1060-3425/97 $10.00 (c) 1997 IEEE

GIOP version, and the length (in octets) of the message following the header. A complete GIOP message consists of the GIOP message header, a header of the concrete message type (e.g. a Request Header) and in particular message types, a body. The header of the concrete message type contains information such as request/reply ids or operation names. The body consists of the message‘s invocation or reply parameters. The GIOP may be implemented on several transport protocols that match a set of assumptions: it should be connection-oriented, reliable, provide notification of disorderly connection loss, the transport can be viewed as a byte steam, and the model for initiating connections of the particular transport protocol can be mapped onto the TCP/IP general connection model (namely the `listen,connect,accept/reject` steps).

The connection management of the GIOP identifies client/server roles with their particular behaviour and describes possible ways to close and multiplex connections. The GIOP supports flexible object migration and location services. A given transport address must not correspond to a specific ORB component, such as an object adapter, object server process, etc. It merely implies the existence of an entry point at which a connection may be opened, and to which requests may be sent. This entry point, the server, can act in different ways in respect to a particular object reference. E.g. the server may be able to accept the request directly and return replies or the server may act as a location service providing new addresses to which requests may be sent.

The Internet Inter-ORB Protocol

The mapping of GIOP message transfer to a specific TCP/IP connection is called the Internet Inter-ORB Protocol (IIOP). The GIOP and IIOP support protocol-level ORB interoperability. The exchanged messages are encoded in the Common Data Representation (CDR) format. The TCP/IP address of the servers are published in IORs. Therefore IIOP defines IIOP IOR profiles:

```
module IIOP {
  struct Version {
    charmajor;
    charminor;
  };
  struct ProfileBody {
    Versioniiop_version;
    stringhost;
    unsigned shortport;
    sequence <octet>object_key;
  };
};
```

An instance of the `IIOP::ProfileBody` type is marshalled into an encapsulated octet stream (`sequence <octet>`). This stream becomes the `profile_data` member of the `IOR::TaggedProfile` structure representing the IIOP profile in an IOR, and the tag has the value `TAG_INTERNET_IOP` see section Interoperable Object References - IORs. A TCP/IP address consists of an IP host address and a TCP port number. The `object_key` member is a value supplied by the agent producing the IOR. This value will be used in request messages to identify the object to which the request is directed, and should not be modified by any client.

The clients initiate a connection with the address specified in the IOR. The listening server may accept or reject the connection. Once a connection is accepted clients may send `Request`, `LocateRequest`, or `CancelRequest` messages by writing to its TCP/IP socket. The server may send `Reply`, `LocateReply` and `CloseConnection` messages by writing to its connection. After a `CloseConnection` message, both the client and the server must close the TCP/IP connection.

3. CORBA 2 Interoperability - Practical Part

After the more theoretical considerations of CORBA communication and CORBA Services, we were forced to think realistically to realise the DPE for our distributed telecommunication application for a multimedia communication service (T-MMCS). At the end of 1995 we decided to use Smalltalk and C++ as implementation languages and therefore we began to look for some CORBA realisations supporting Smalltalk and C++. We wanted to use the basic CORBA communication facilities as well as some CORBA Services. We wanted to use a heterogeneous system and therefore had to use InterORB communication (IIOP). We decided to use Orbix for the C++ side since we had some experiences with Orbix and Orbix announced support for CORBA 2 (IIOP) in the first quarter of 1996. For the Smalltalk we decided to use HPDST 5.0 since it supported IIOP and implemented some CORBA Services.

As a startingpoint, we had a lot of IDL[1] specifications describing the interfaces of the objects which were supposed to communicate. For the first approach we decided to use two CORBA Services, the Naming Service and the Event Service. Both had to be used for the Smalltalk part as well as for the C++ components.

3.1 From the IDL spec to the implementation

Starting with the IDL specifications means that you have to map the IDL constructs to language specific con-

1. IDL = Interface Definition Language for CORBA 2 objects. Precisely spoken we had TINA-C ODL [11] (Object definition Language) specifications. ODL is a superset of CORBA IDL.

structs (C++ or Smalltalk) and to implement the semantics of the objects that should be represented by the IDL specs. For C++ this is supported by the Orbix system that generates C++ header files from the IDL description. HPDST does not have such a generator that creates Smalltalk skeletons from the IDL spec. HPDST is designed to start with a Smalltalk program that should be distributed. Using this Smalltalk program you can use a generator that will generate the IDL description of the interfaces for distributing the program. But this could not be used in our case - we had the IDL specs and needed the implementations. Therefore we had to write the generator of skeletons on our own or to do the mapping by hand. We created a specialized generator for our needs.

After we had the implementations for the objects specified in IDL on the C++ and the Smalltalk side our first test showed that Orbix did not implement the IIOP correctly. We had to do a lot of beta testing, bug fixing and find work arounds. Some small errors also occurred on the HPDST side.

3.2 The Lifetime of Object References

One major problem that occurred on both sides concerned the lifetime of object references. How long will a reference to an object be valid? On the C++ (Orbix) side we have a daemon that receives all requests to objects located at its side and starts a program that implements the object addressed. After execution this object may close down. Will references to objects inside other objects and objects which are created and "exported" by the requested operation be valid after the close down? The same question arises at the Smalltalk side in respect to objects created in an image for which references are "exported" using CORBA communication to objects outside the image. Will they be valid after the image is saved, closed and restarted?

Within the CORBA 2.0 specifications you can find the following statements concerning the lifetime of object references:

Objects (or, more precisely object references) are made persistent by the BOA and the ORB Core, in that a client that has an object reference can use it at any time without warning, even if the implementation has been deactivated or the system has been restarted. Although the ORB Core and BOA maintain the persistence of object references, the implementation must participate in keeping any data outside the ORB Core and BOA persistent.

With Orbix the only way to ensure the durability object references was not to close the implementation in which it is contained. That means have it running static.

For HPDST we have to create a static configuration of the ORBs (images), this means statically assign portnumbers etc. in the configuration setup for each HPDST image. In this case the references to objects in the image will be valid after saving, closing and restarting the image.

For the HPDST we are working on a solution which integrates Objectstore [15], an OODB, to ensure persistence of objects and object references without saving the image. Even "killing" the image should not destroy the object reference and the data defined as "persistent".

3.3 Using CORBA Services

For the first approach we decided to use the Naming Service and the Event Service supplied by HPDST. To create objects in the HPDST image from an object situated in the C++ or another HPDST image, we have to use parts of the Lifecycle Service (the Factories Finders and Factories).

The **Naming Service** offers the ability to bind a name to an object relative to a naming context. This name can be passed to other objects. Objects can address the naming service and resolve the name in a given context. This will "retrieve" the named object for later use. Since the interface to the Naming Service provided by HPDST has an IDL specification as a CORBA Service, it can also be used through this interface by Orbix. This is done in our DPE to provide a common naming service and theseby provide access to the objects registered there. After working around some minor errors, this service works fine.

The **Event Service** should be used to create objects (event channels) that accept notifications, stores them intermediately and forwards (push) them to consumers of the notifications that are connected to the event channel object. Another supported model allows the event channel to store the notifications for the consumers connected to them until they pull the notification from the channel. An event channel can support multiple push and pull consumers connected to it at the same time. One event channel can also have more than one object that supplies events to it. The concept of event channels should decouple suppliers and consumers of notifications. By providing different implementations of event channels, different policies for delivering events can be realized (e.g. only one consumer should receive a notification or it should be reliably delivered to all consumers etc.). By default only the policy that all consumers which are connected to the event channel receive the event is implemented in HPDST.

One problem arose using the push model, which involves having a consumer get the notification directly forwarded as it arrives in the event channel. The implementation in HPDST will not decouple the sender and the receiver of the notification. The sender pushing the

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

IEEE COMPUTER SOCIETY

1060-3425/97 $10.00 (c) 1997 IEEE

event to the event channel will be blocked until the consumer connected to the event channel has received and processed the event. This blocking was not an effect we expected for the CORBA event service. The pull consumer will not block the sender of an event to the channel until it pulls the event from the channel. CORBA in its introduction of the Services ("Summary of key features") stresses the "asynchronous event (decoupled event supplier and consumer)". However, in the "Event Service Specification" the part on "Event Channels" only says that "An event channel *can* provide asynchronous communication of event data between suppliers and consumers.".

These "inconsistencies" makes it possible for HPDST to be a CORBA compliant service but unpredictable for the user in respect to how the service will work.

What about the **other CORBA Services**?

We did not use the other CORBA Services from the start for two reasons. Some of these services require an implementation on all the platforms (C++ and Smalltalk) that want to use the service, and we only had an implementation in HPDST. The services were also either not implemented completely. New implementations are planned for HPDST and Orbix and for other ORBs, so the world of CORBA communication is changing and we plan to use other CORBA Services to support the transparencies sketched in section 1.3.

4. Conclusion

The basic CORBA Communication mechanism and the CORBA Services seem to be a good approach for a distributed processing environment to achieve distribution transparencies. Different kinds of transparencies are supported by the different CORBA services. One of the main problems with the CORBA communication facilities is the fidelity of object references. This problem must be solved within the ORBs and Object Adaptors. Another problem is that the IDL only specifies the syntactical interface of an object and not the behaviour of the object. It could be a problem in the future that there are several implementations of the same interface but with different behaviour. Upcomming languages like Java open an interesting field for the CORBA communication area since we can not only have the mobility of an object instance, its status represented by its internal data, but also its behaviour, represented by its methods. This will be an interesting point - using dynamically changing Interface and Implementation Repositories and the exchange of implementations e.g. encoded in Java.

5. References

[1] M. Altenhofen, J. Dittrich, R. Hammerschmidt, T. Käppner, C. Kruschel, A. Kückes, T. Steinig, "The BERKOM Multimedia Collaboration Service" in Proc. of the First ACM International Conference on Multimedia, Anaheim, California, 1.-6. August 1993

[2] H. Berndt, L.A. de la Fuente, P. Graubmann, "Service and Management Architecture in TINA-C" in Proc. of the TINA'95 Conference, February 1995, Melbourne, Australia

[3] K.-P. Eckert (Ed.), "The Object Management Group – OMG – Concepts, Architectures and Experiences, Recommendations for the Migration of Multimedia Teleservices to the world of Distributed Object Systems", GMD-FOKUS, Deliverable of the BERKOM-Project MMTS, Release 1.0, January 1995

[4] K. Hofrichter, "GLASS: A Multimedia Presentation System utilizing MHEG Objects" in Proc. of the 1st International Workshop on High Speed Networks and Open Distributed Platforms, 12.-15. June 1995, St. Petersburg, Russia, 1995

[5] J. Insulander, P. Schoo, I. Tönnby, S. Trigila, "An Architectural Approach to Integrated Service Engineering for an Open Telecommunication Service Market" in Proc. of the International RACE IS&N Conference on Intelligence in Broadband Services and Networks, November 23-25, 1993, Paris, France

[6] ITU-T | ISO/IEC Recommendation X.901 | International Standard 10746-1, "ODP Reference Model: Overview", June 1995

[7] G. Nilsson, F. Dupuy, M. Chapman, "An Overview of the Telecommunication Information Networking Architecture" in Proc. of the TINA'95 Conference, February 1995, Melbourne, Australia

[8] OMG, "The Common Object Request Broker 2.0: Architecture and Specification", July 1995

[9] OMG, "CORBA Services: Common Object Services Specification", March 31, 1995

[10] TINA-C, "Overall Concepts and Principles of TINA", TINA Baseline, TB_MDC.018_1.0_94, February 1995.

[11] TINA-C, "TINA Object Definition Language (TINA - ODL) MANUAL", Version 1.3, June 1995

[12] IONA Technologies, "The Orbix Architecture", August 1993.

[13] Hewlett-Packard, "HP Distributed Smalltalk 5.0, Programmer's Reference Guide", October 1995.

[14] J. Dittrich, P. Schoo, "Experiences with the Implementation of a Multimedia Collaboration Service for an Environment of Distributed Platforms", submitted to the HICSS-30)

[15] ObjectStore: http://www.odi.com/prodinfo/os/techovrwv.html

Proceedings of The Thirtieth Annual Hawwaii International Conference
on System Sciences ISBN 0-8186-7862-3/97 $17.00 © 1997 IEEE

1060-3425/97 $10.00 (c) 1997 IEEE