APM

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE • CB3 0RD UNITED KINGDOM
+44 1223 515010 • Fax: +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

ANSA Phase III

An Introduction to DIMMA

Ian Macmillan

Abstract

The aim of this document is to introduce the facilities provided by DIMMA , position it with respect to CORBA and highlight the benefits of using DIMMA over those of a standard commercial ORB.

DIMMA is a CORBA compliant ORB with extensions for supporting soft real-time and multi-media applications. It provides applications with control over their use of resources in order to achieve specified quality of service. Flow interfaces are provided to transfer continuous flows of multi-media data such as video and audio.

DIMMA provides these facilities ahead of commercial ORB offerings and allows the exploration of soft real-time distributed application space: applications such as video on demand.

APM.1995.02 **Approved** 30th September 1997

Technical Report

Distribution:
Supersedes:
Superseded by:

Copyright © 1997 APM Limited
The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.

An Introduction to DIMMA

An Introduction to DIMMA

Ian Macmillan

APM.1995.02

30th September 1997

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by APM Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

APM Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(01223) 515010
INTERNATIONAL	+44 1223 515010
FAX	+44 1223 359779
E-MAIL	apm@ansa.co.uk

Copyright © 1997 APM Limited
The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.

APM Limited takes no responsibility for the consequences of errors or omissions in this Report, nor for any damages resulting from the application of the ideas expressed herein.

Contents

3	**1**	**Introduction**
3	1.1	Overview
3	1.2	How DIMMA is different from CORBA
4	1.2.1	What problem does DIMMA address
5	1.2.2	How QoS is specified
5	1.3	Who should use DIMMA
6	**2**	**Structure**
6	2.1	Overview
8	**3**	**Functionality**
8	3.1	Introduction
8	3.2	Jet
8	3.2.1	Flow Interfaces
9	3.3	ODP
9	3.4	Binding
10	3.5	QoS and Resource Management
10	3.6	Threading and Locking
11	3.6.1	Client threading
11	3.6.2	Server threading
11	3.6.3	Locking
12	3.7	Trader

1 Introduction

1.1 Overview

DIMMA, an acronym for Distributed Interactive Multi-Media Architecture, is used to describe both the architecture and the example implementation. In this document, DIMMA is intended to refer to the implementation.

DIMMA is an open distributed processing (ODP) platform whose purpose is to facilitate the production of distributed applications. DIMMA consists of a layer of distribution engineering (middleware) running over a standard operating system, together with tools to interface applications to this engineering.

The most popular commercial ODP platform is the Common Object Request Broker Architecture (CORBA) whose definition is managed by the Object Management Group (OMG), see [OMG 95]. In recognition of this popularity, DIMMA supports a CORBA compliant API or *personality*, so that CORBA applications may be easily ported to or from the DIMMA platform.

1.2 How DIMMA is different from CORBA

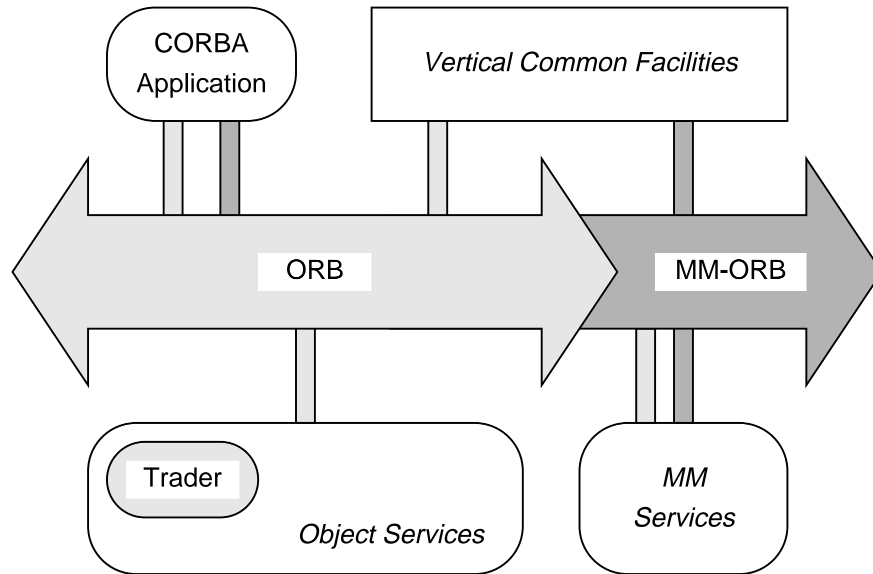
CORBA was designed to offer applications maximum transparency to the concerns of distribution and to offer portability across a wide variety of operating systems. As a consequence, CORBA does not address the needs of an increasingly large class of applications that must deliver their results within a particular time scale (soft real-time), nor does it address the need for communication of continuous flows of data such as audio or video.

DIMMA is designed to explore how the needs of such applications may be met in the context of an ODP platform. To this end it provides applications with control over their allocation and use of resources through quality of service (QoS) parameters, and supports multi-media flows through flow interfaces. These facilities may be regarded as extensions to CORBA as shown in figure 1.1.

In contrast to some rather monolithic CORBA implementations, a major feature of the DIMMA design is its open flexible component architecture. DIMMA is very highly configurable: it supports multiple protocols - new protocol implementations may be dynamically incorporated into an application at run-time; and QoS parameters allow an application programmer to exercise a high level of control over the configuration of the components that are used to provide an engineering channel.

DIMMA is not intended to be a fully functional commercial ORB: it implements the core ORB functionality plus real-time and multi-media extensions. It is not supplied with any Common Facilities and provides only one Object Service in the form of a primitive trader. DIMMA provides the shaded areas in 1.1.

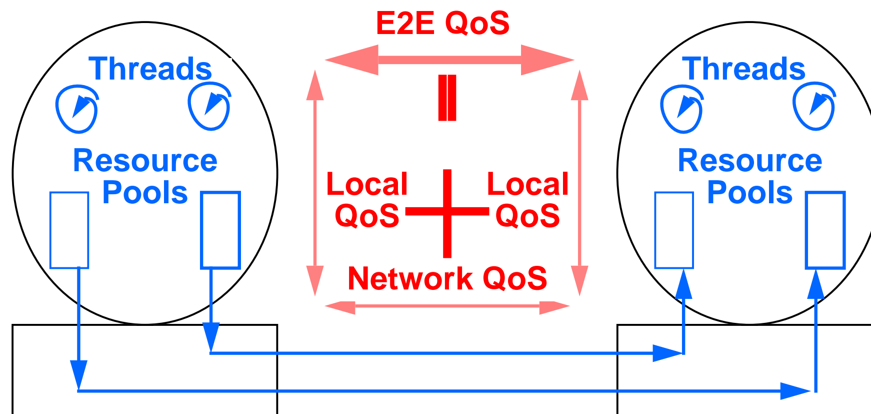
Figure 1.1: Extensions to CORBA



1.2.1 What problem does DIMMA address

DIMMA addresses the needs of the class of applications that require soft real-time QoS guarantees. These guarantees must be delivered end to end at the application. Hence each intervening layer, such as the distribution infrastructure, must provide QoS guarantees that together will result in what the application needs. This is illustrated in figure 1.2 and indicates that applications must be able to reserve their resources, such as threads, and control the behaviour of the underlying infrastructure, e.g the level of communications multiplexing.

Figure 1.2: End to end QoS



In practice, guarantee is too strong a term in that distributed systems are inherently unpredictable and hard guarantees would be difficult to achieve.

Fortunately, many applications do not require rigorous guarantees but are prepared to accept some form of bounded behaviour which is more easily achieved. Furthermore, some applications are able to adapt to variations in the underlying QoS.

DIMMA also addresses the requirements of applications that require an open flexible architecture where the ability to add new protocol implementations to the basic ORB is a requirement, for example to provide interoperability with existing legacy systems.

1.2.2 How QoS is specified

In order to provide any kind of bounded QoS, it must be possible for an application to communicate its requirement to the underlying infrastructure, both distribution layer and host operating system.

DIMMA supports a resource reservation model and allocates resources according to the specified QoS when an application establishes a binding, e.g. when a client binds to a server. Resources are reserved for all parts of the underlying channel, e.g. communications resources, buffers, tasks, etc. The binding model used by DIMMA to support the specification of QoS, is known as explicit binding.

1.3 Who should use DIMMA

DIMMA is intended primarily as an experimental vehicle. The focus is on using DIMMA to identify the needs of multi-media and real-time distributed applications in terms of proposed core ORB facilities and to prototype the resultant ideas.

Although DIMMA will run ‘out of the box’, it is anticipated that it will be of greatest value to those who wish to customise the core ORB. The internal structure is very flexible and the components are built according to a set of well defined frameworks. For example, new protocols may be easily added using the DIMMA protocol framework which facilitates the reuse of layered modules.

The explicit binding model, together with the flexible resource reservation mechanisms allow a considerable range of applications to be built: applications that are not possible on todays standard commercial ORBs. This should be of interest to those involved in telecommunications applications such as video on demand.

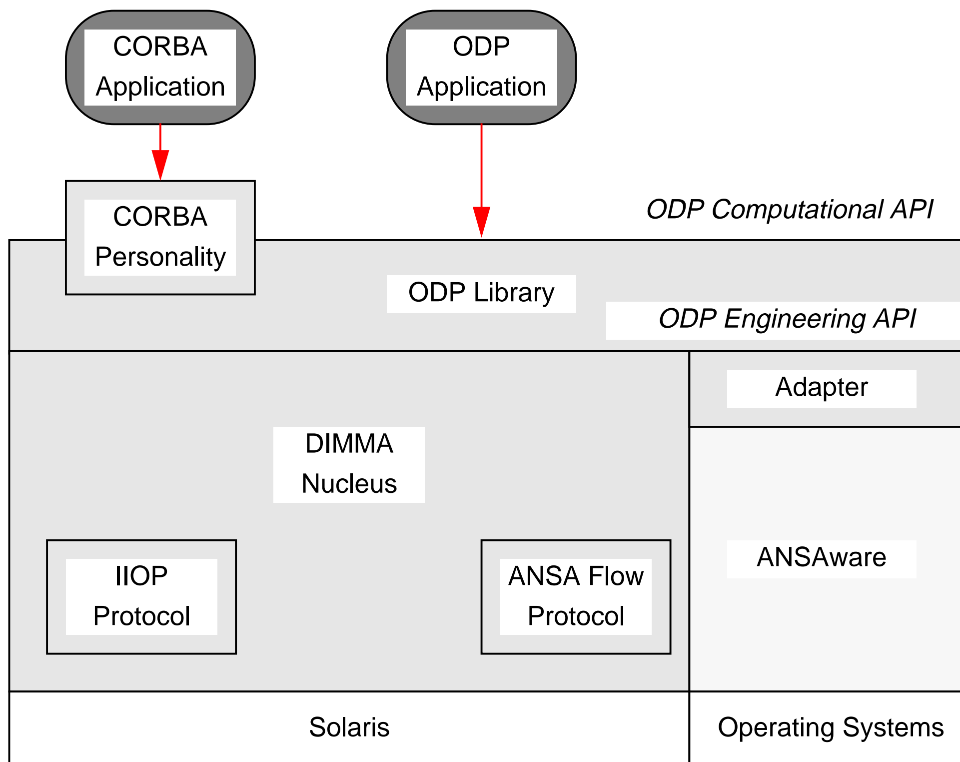
2 Structure

2.1 Overview

DIMMA is constructed as a set of small components which may be combined in many different ways to suite the diverse needs of applications and in recognition of that fact that the real world needs ORBs that are high performance, downsizeable and scalable. In this sense, DIMMA may be regarded as a microkernel ORB.

In order to describe the functionality provided by DIMMA, the total set of modules are considered to be subdivided into groups and arranged in layers; this is also reflected in the source structure. This layering is depicted in figure 2.1

Figure 2.1: Structure of DIMMA



DIMMA provides two application programming interfaces (APIs): a CORBA extended subset called Jet and a proprietary one based on the ODP-RM (see [ISO/IEC 95]) concepts (ODP Computational API). Jet is implemented as a *personality* built on top of the ODP facilities and hence shares a number of the former's features.

Both APIs are mapped onto a common ODP Engineering API by the ODP Library in order to facilitate hosting on different ORBs (or Nuclei in ODP terminology). The DIMMA nucleus supports the ODP Engineering API directly, whilst adapters must be provided for other ORBs. To date only one adapter has been produced and this was to allow applications written using Jet or ODP to communicate over ANSAware.

The DIMMA nucleus provides the distribution engineering apparatus such as binders and communications protocols. Currently two protocols are provided: the CORBA Internet Interoperability Protocol (IIOP) for interoperability with other vendors ORBs and a proprietary protocol (ANSA Flow), optimised for transporting multi-media flows.

The DIMMA nucleus must be hosted on an operating system able to provide the necessary soft real-time facilities, for example Posix threads [POSIX]. The current release of DIMMA runs over Solaris 2.5.

3 Functionality

3.1 Introduction

The purpose of this chapter is to give an idea of the facilities that are provided and in the case of proprietary extensions, to give an overview of their mode of use. It is not intended to describe all the facilities of DIMMA in detail: such descriptions may be found in other documents which are referenced in the sections below.

3.2 Jet

Jet supports most of the CORBA C++ language mapping and has an associated IDL compiler. It does not support any CORBA-style repositories. It is intended for application writers who are already familiar with CORBA and who may wish to port their applications between DIMMA and another vendors ORB. For details on how to write an application using the Jet API, see [APM.1987].

3.2.1 Flow Interfaces

There is a class of applications for which the operational RPC mechanism is inappropriate. These applications deal naturally in continuous flows of information rather than discrete request/reply exchanges. Examples include the flow of audio or video information in a multimedia application, or the continuous flow of periodic sensor readings in a process control application.

A flow has a distinct type and an associated direction with respect to the binding, i.e. video information might flow out of a producer binding associated with a camera and into a consumer binding associated with a TV monitor on which the output of the camera is to be displayed. It follows that flow interface types exist in pairs which are related by the reversal of the flow.

The type of flow is characterised by the set of possible frames that it can support. For example, a video flow might be able to carry both MPEG and JPEG frames.

A flow interface is modelled in Jet IDL in terms of oneway operations and will result in the generation of both producer and consumer components, in an analogous way to client and server components generated from an operational interface. The main difference is that the operations are oneway and data is unidirectional with respect to the binding. The operations within a flow interface correspond to frame types and these are further described by the parameters of the operation.

For example, a simple video flow consisting of a single frame type (Frame1) consisting of a frame number and the video image data (image), could be described in IDL as follows:

```
flow Video
{
void Frame1(in long frame_no, in string image);
};
```

Note that both parameters are described as input (in) and that the operation returns no result (void type).

The ODP standard defines an additional concept called *stream* which is described as a set of unidirectional flows, e.g. a TV stream might be considered a logical entity consisting of an audio and video flow. DIMMA does not implement the stream concept directly, although in principal, a stream binding could be constructed by an application from a set of flow bindings.

Although flows appear to be similar to operational interfaces in IDL, they are distinct entities and a flow cannot inherit from an interface, nor can an interface inherit from a flow.

3.3 ODP

The ODP Computational API is a proprietary API based on the concepts of the ODP-RM (see [ISO/IEC 95]). It provides the minimum facilities necessary for writing distributed applications but also affords considerable flexibility. For example, unlike CORBA, an ODP application object can export multiple interfaces.

It is intended primarily for hosting *personalities* such as Jet rather than for writing applications. For this reason there is no IDL compiler provided and hence an ODP application writer must provide their own stubs.

3.4 Binding

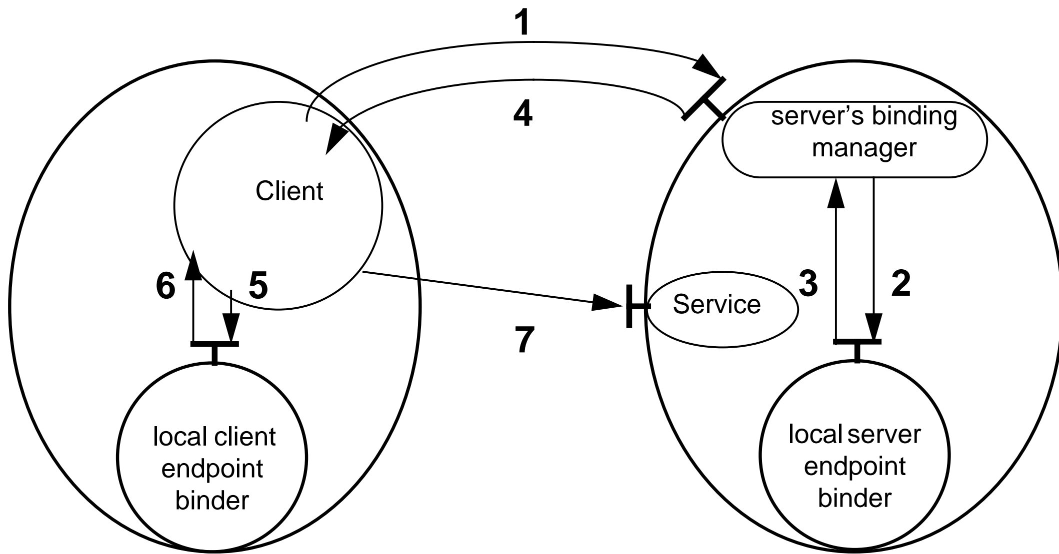
DIMMA supports both implicit and explicit binding models. The former is the model adopted by the majority of ORBs and provides maximum transparency to the application in terms of hiding irrelevant engineering details. However, there is a trade-off: transparency implies little or no control over the QoS of the binding that is established. For multi-media and real-time applications this is unacceptable and an alternative model is required.

To meet the needs of multi-media and real-time applications, DIMMA provides a model of explicit binding. Explicit binding allows both the QoS and the time of binding to be controlled and hence allows resource reservation. The downside is the increased complexity of the mechanism required to establish the binding.

An explicit binding is accomplished in several stages and is bootstrapped using the implicit binding mechanism. An application wishing to offer a service must do so via a binding manager which offers its own interface as a proxy for the real service. The binding manager is responsible for placing each parties local explicit binder in communication with one another. These in turn will set up the local bindings with the specified QoS and establish the network connection.

This procedure is illustrated in figure 3.1 and is described in more detail in [APM.1392].

Figure 3.1: Explicit Binding



The implicit binding in DIMMA is implemented by Binder objects which implement a predefined binding policy, and make use of default QoS parameters. In keeping with the DIMMA component philosophy, the implicit binders may be replaced by application specific implicit binders.

3.5 QoS and Resource Management

DIMMA provides for application control and management of resources through the use of QoS parameters.

DIMMA QoS parameters are specified in terms of engineering attributes such as Buffers, Threads, Communication EndPoints *etc*. Mapping between application specific QoS parameters (e.g. jitter, frame rate *etc*.) and Engineering QoS parameters remains the responsibility of the application in DIMMA 2.0

The DIMMA QoS infrastructure allows the application explicit control over the resource allocation policy to be used within an given engineering channel. For example, buffer allocation may be defined to be 'on demand', or a set of buffers may be preallocated for the channel's exclusive use.

An application may also make explicit decisions about the concurrency model to be used by the channel: maximum flexibility may be achieved by allowing thread multiplexing on the channel; or high performance may be achieved by prescription of a dedicated threading model[1].

3.6 Threading and Locking

DIMMA offers both single-threaded or multi-threaded operation. The latter is implemented internally using Posix threads. Since CORBA does not define

1. The startling performance improvements that DIMMA can provide through judicious choice of QoS parameters is illustrated in [APM.2046]

any standards for multi-threaded operation, DIMMA provides its own interfaces to control threading and locking.

The DIMMA model of threading provides two abstractions: Threads and Tasks. Threads are a unit of potential concurrency and are scheduled over tasks by DIMMA. Tasks represent a unit of real concurrency and are implemented as Posix threads [POSIX] (bound to a lightweight process on Solaris). Tasks are scheduled pre-emptively by the underlying operating system and it is the applications responsibility to ensure that access to data shared between tasks is properly synchronised.

3.6.1 Client threading

Client objects directly control their concurrency by creating tasks appropriately. It is unlikely that a client object would use threads rather than tasks. The interface to DIMMA tasking borrows heavily from Java. A DIMMA task executes an instance of a Runnable interface, the latter being an abstract class from which a concrete class should inherit. The Runnable interface defines a single method called Run which will be the entrypoint for the task.

A simple example of its use is as follows:

```
#include <Runnable.hh>

class MyRunnable : pubic Runnable
{
     Runnable::status_t Run();
     ...
};

main()
{
     Runnable * runnable = new MyRunnable;
     Task * MyTask = new Task;
     MyTask->Install(MyRunnable);
     MyTask->Start();
     (void)MyTask->Join();
     delete MyTask;
}
```

3.6.2 Server threading

An server object shares the same engineering for threading as that provided for client objects but the use is typically different. The reason for the difference is that server interfaces are normally upcalled by a nucleus task, rather than being executed by a task provided by the server object. Control over concurrency in the server is specified through QoS attributes when an interface is created using explicit binding. It is possible to specify whether the operation should execute in the nucleus task, a new server task or whether a new thread should be queued for subsequent execution by an associated task.

3.6.3 Locking

DIMMA provides mutual exclusion locks (mutexes) and condition variables, the latter being used to wait for a condition to occur without consuming CPU. These may be accessed through the Mutex and Condition classes.

In addition a Java style synchronisation interface is provided which uses the same mechanisms but in a more natural and less error-prone manner. A class Synchronisable may be inherited by any class wishing to perform synchronisation. This class effectively provides an object level lock, a class level lock and an associated condition variable for coordinating access to the object's state.

An object may protect itself using the following construct:

```
{
    SynchronisedObject(this);
    ...
    protected statements;
    ...
}
```

Note the use of braces the scope the extent of the Synchronised region.

3.7 Trader

DIMMA provides a primitive trader facility based on shared file access, e.g. NFS. An interface being exported is stringified and written to a file named after the interface and located in the directory from which the server was started.

A client importing an interface, locates the file based on the interface name, reads the content and converts the stringified interface reference back into its internal form. The interface reference returned may subsequently be used to invoke the associated server.

How to access the trader from the Jet API is described in [APM.1987].

References

[APM.1392]

Otway D., *The ANSA Binding Model*; **APM.1392**, APM Ltd., Cambridge U.K., Jan 1995.

[APM.1987]

Howarth N., *Programming in Jet*; **APM.1987**, APM Ltd., Cambridge U.K., May 1997.

[APM.2046]

Dimma Team, *Dimma Performance Analysis;* **APM.2046,** APM Ltd., Cambridge U.K., August 1997.

[ISO/IEC 95]

ISO/IEC 10746-3, *Reference Model of Open Distributed Processing*, Jan 1995.

[OMG 95]

The Object Management Group, *The Common Object Request Broker: Architecture and Specification*; **Revision 2**, OMG Headquarters, 492 Old Connecticut Path, Framingham, MA 01701, Jul 1995.

[POSIX]

POSIX, *IEEE POSIX Std 10003.4a*, Sept 1992.