Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

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

ANSA Phase III

DIMMA Architecture Overview

Andrew Herbert, Matthew Faupel

Abstract

This document reproduced the original specification of the ANSA Phase III implementation of the Distributed Interactive Multimedia Architecture (DIMMA). It provides a contet for interpreting the final DIMMA architecture reports, technical reports and software prototypes.

APM.2065.00.01 **Draft** 30th September 1997

Technical Report

Distribution:
Supersedes:
Superseded by:

DIMMA Architecture Overview

DIMMA Architecture Overview

Andrew Herbert, Matthew Faupel

APM.2065.00.01

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

Contents

1	**1**	**Introduction**
1	1.1	Overview
1	1.2	Document organization
2	**2**	**Objectives for DIMMA**
2	2.1	Business case
2	2.1.1	Drivers
2	2.1.2	Challenges
2	2.2	Scope
3	2.3	Goals
3	2.3.1	Architecture
4	2.3.2	Programming model
4	2.3.3	Engineering model
5	2.3.4	Implementation
5	2.4	Deliverables
6	**3**	**Programming Model**
6	3.1	Plan
6	3.1.1	A: logical programming model
7	3.1.2	B: CORBA implementation
7	3.1.3	C: CORBA extension
7	3.2	Components
7	3.2.1	PL and C++
7	3.2.2	PL tools
7	3.2.3	Runtime
7	3.2.4	CORBA and C++
8	3.2.5	CORBA IDL extension
8	3.2.6	CORBA C++ tools extension
8	3.3	Documentation
9	**4**	**Engineering Model**
9	4.1	Plan
9	4.2	Major components
9	4.2.1	Stubs
9	4.2.2	Runtime
9	4.2.3	Channels
10	4.2.4	Sessions
10	4.2.5	Generic communication stack
10	4.2.6	Execution protocols
11	4.2.7	Message passing service
11	4.2.8	QoS mapper
11	4.2.9	Implicit binder
11	4.2.10	Explicit binders

11 4.2.11 Threads
11 4.2.12 Tasks
11 4.2.13 Entries
11 4.2.14 Buffers
11 4.2.15 Scheduler
12 4.2.16 Events handler
12 4.3 Documentation

1 Introduction

1.1 Overview

This document is essentailly the initial description of the goals for the development of the ANSA Distributed Interactive Multimedia Architecture (DIMMA) and associated software prototypes. It provides a context for the set of architecture and technical reports describing what was accomplished in the course of the project..

1.2 Document organization

The document is structured as follows:

Chapter 2 identifies the major objectives and goals for the architecture.

Chapter 3 outlines the major components of the DIMMA programming model and the plan for their development.

Chapter 4 outlines the major components of the DIMMA engineering model and the plan for their development.

2 Objectives for DIMMA

2.1 Business case

2.1.1 Drivers

A market for interactive, wide area access to multi-media information and services is being created by technological developments including

- public access to networked informations services, and electronic commerce via “the information superhighway”
- the increasing use of multi-media presentation and interaction in education, training, entertainment and games
- the deployment of broadband communication networks, leading to increasing bandwidth into the office and home
- the use of open systems technology and networking in embedded systems (e.g., telecommunications, consumer electronics and manufacturing automation)
- the development of broadband network interfaces to personal computers, workstations and servers enabling them to support distributed multi-media interactive services.

2.1.2 Challenges

Software is lagging behind the development of basic technology and will hinder widespread exploitation of the drivers until

- dynamic approaches to configuration and management of information and information services replace current static, pre-planned, over managed approaches
- transparent interworking is possible between the different distribution platforms used for the desktop (e.g. OLE 2, CAIRO), on workstations (e.g. CORBA), on servers (e.g. DCE) and in information networking (e.g. INA).
- it is possible for a computer to interact with other computers via broadband communication networks (both wide area and local area) to set up service-oriented interactive sessions (e.g., for computer supported cooperative working, multi-party commercial transactions)
- it is possible for applications to coordinate different media flows across the network and local processing resources to meet users’ quality of service expectations.

2.2 Scope

The DIMMA addresses service management, services binding and services Quality of Service (QoS) management at a level of abstraction consistent with

the application programming interfaces found in current distributed object computing systems such as ANSAware, Microsoft's OLE 2, OMG's CORBA standard and Bellcore's INA.

DIMMA will cover

- the extensions that are needed to current distributed computing object models (e.g. stream interfaces), and their manifestation in applications programming systems
- the additions and extensions that are needed to supporting services for distributed services management (e.g. explicit binding operations)
- the extensions needed to current distributed object computing infrastructures to enable interworking between them (e.g. support for multi-protocol ORBs)
- the extensions needed to current distributed object computing infrastructures to enable fine grained control and monitoring of resources to give integrity to quality of service guarantees
- the extensions needed to support services for distributed services management and fine grained control and monitoring of resources in an overall distributed system.

The work will use the ODP object model, and the work to date in ANSA on real-time and multimedia computing, quality of service management and performance management as a baseline.

The planned work will

- detail and animate the architecture outlined above
- develop prototype technology that shows what has to be added to current distributed object computing systems to meet the above requirements
- identify strategies for enhancing the manageability, performance and predictability of current distributed object systems and their supporting operating systems.

We plan an incremental stream of architecture and prototype technology primarily directed at sponsors' broadband interactive multi-media development projects.

The planned work consists of two areas: a programming model (PM) and an engineering model (EM). The PM provides a clean high level interface for the application programmer to use the DIMMA. The EM provides the mechanisms that enable the execution of the distributed services specified by the PM.

2.3 Goals

The DIMMA design and implementation allows a complete evaluation and validation of the progress made in ANSA Phase 3 performance architecture work [APM.1108], [APM.1151], [APM.1239] and [APM.1222].

2.3.1 Architecture

- follow ANSA principles
- maximum compatibility with RM-ODP

- maximum alignment with CORBA
- clear separation of programming model from engineering model.

2.3.2 Programming model

- extend ANSA Computational Model
  - — provide a computationally complete set of constructs for the full and efficient support of distribution
  - — strong type checking
  - — high level abstraction of engineering details
  - — enable local optimization
- access and location transparency
- explicit and implicit binding
- synchronous and asynchronous computing
- operational interfaces and stream/signal interfaces
- selective resource transparency
  - — controlled scheduling
  - — controlled communication multiplexing
- declarative specification of QoS directives

2.3.3 Engineering model

- generic framework (interface) enabling different implementations and implementation tradeoffs are possible
  - — modular - easy to replace alternative components
  - — extensible - easy to add extended functionality
  - — scale up (for large applications of many clients and servers) and scale down (for embedded applications which have limited resources)
- scalable and resource-efficient implicit binding
- QoS driven explicit binding
- generic communication architecture for multiple protocol stacks, addressing schemes and communication models
- different execution protocols for different style of object interactions
- generic QoS specification, conformance check, negotiation and monitoring
- resource separation and independent scheduling
- both real-time and time-sharing scheduling
- allow to map onto any suitable real-time and multimedia technology
- end-to-end communications management
- maintain only minimum functionality to enhance system portability and applicability
- open to alternative APIs (e.g. specialized languages, preprocessor, library, CORBA etc.)

2.3.4 Implementation

- maintainable - a common coding/documentation standard
- using object-oriented design and particularly C++ language for programming
- maximum reuse of CORBA implementation(s)
- maximum reuse of ANSAware
- good raw performance
- use industry standards wherever possible (CORBA, POSIX, ANSI, ITU etc.)

2.4 Deliverables

The deliverables from DIMMA are

- architecture summary technical reports
- consultancy
- prototype software
- illustrative working examples.

The work will also generate

- inputs to CORBA evolution
- defect reports to RM-ODP
- baseline for future ODP component standards

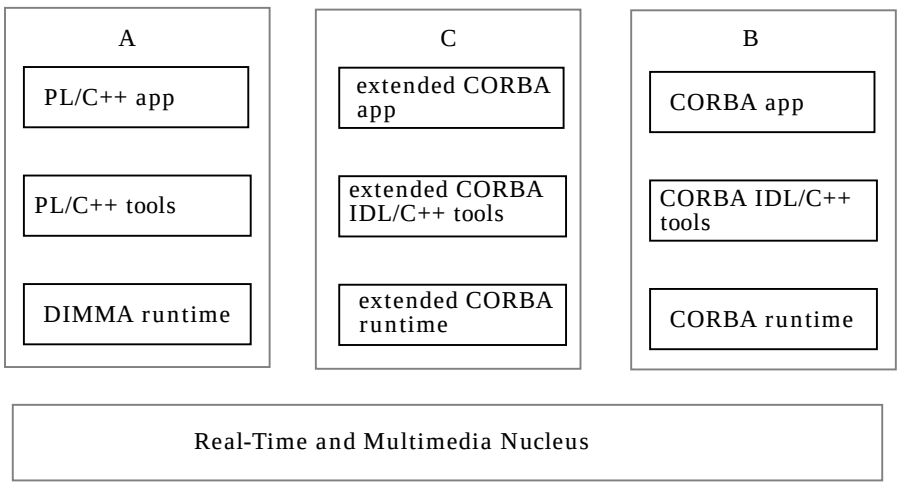
3 Programming Model

The programming model provides a clean high level interface for the applications programmer to use the DIMMA.

3.1 Plan

The PM work can be divided into three aspects as shown in figure 3.1.

Figure 3.1: Aspects of the Programming Model



3.1.1 A: logical programming model

See box A in figure 3.1. This work provides a full and consistent coverage of the real-time and multimedia computing using ODP/ANSA architecture as the baseline. It has three parts:

- functional design: understand the required functionality in terms of abstractions and structures. To achieve a high-level view, a pseudo language (named PL tentatively, embedded in C++) will be used to write demonstration applications for future elaboration and evaluation of the design choices.
- runtime design: understand the “templates” or runtime classes necessary to support the execution of the functional design. This can be used as a requirement specification for the engineering model.
- implementation: provide compile, check and link tools, e.g. an Abstract Syntax Tree (AST) constructor, type checker, PL preprocessor, the runtime C++ library etc.

3.1.2 B: CORBA implementation

See box B in figure 3.1. This work provides a full implementation of CORBA/C++ based on our real-time and multimedia nucleus to enable interworking between DIMMA and CORBA. It doesn't address real-time and multimedia issues, but rather provides the basis for the box C work. It has two parts:

- IDL/C++ tools: the CORBA IDL compiler and other CORBA ORB services (e.g. the interface repository and the implementation repository etc.)
- CORBA runtime.

Collaborations with one or more CORBA vendor sponsors are required to avoid "reinventing the wheel".

3.1.3 C: CORBA extension

See box C in figure 3.1. This work integrates the work A and B to extend CORBA/C++ with DIMMA concepts and appropriate technology for real-time and multimedia applications. It has three parts:

- IDL extension
- runtime extension
- tools extension

3.2 Components

3.2.1 PL and C++

PL, a pseudo language, is introduced to provide a complete and convenient tool set implementing the ANSA/ODP Computational Model and its extensions for multimedia and real-time processing.

Design should be done in language syntax similar to C++ by providing additional keywords to C++ that allow for the creation of objects, interfaces, operations, stream/signal interfaces, stream operations, explicit binding operations, QoS processing, synchronous computing and real-time programming.

3.2.2 PL tools

An AST will be used to represent PL language level information to allow tools to be re-used across input notations and native targets. An interface type checker is implemented as such a tool.

The AST work should be an extension of the current AST engineering work

3.2.3 Runtime

The differences between the nucleus and multiple distribution tools requires that a different adaptation runtime should be used for each distribution tool set. Three such runtime are required: the DIMMA runtime, CORBA runtime and the extended CORBA runtime.

3.2.4 CORBA and C++

Porting a CORBA/C++ implementation to the DIMMA nucleus.

3.2.5 CORBA IDL extension

Identification of CORBA IDL extensions for streams, explicit binding, QoS specification etc. DIMMA features.

3.2.6 CORBA C++ tools extension

Integration of the DIMMA tools with CORBA/C++ tools.

3.3 Documentation

The following documents should be produced:

- DIMMA programming manual
- DIMMA runtime
- DIMMA tools
- CORBA/C++ and DIMMA

Other relevant documents are [APM.1108], [APM.1239], [APM.1222], [APM.1151] and [APM.TR.031].

4 Engineering Model

The engineering model provides the mechanisms that enable distributed services to be provided in heterogeneous environments. It uses resources in a host computing environment to enable computational objects to be created, managed and executed.

4.1 Plan

The engineering model work can be divided into the following stages:

- functional design: identify the major engineering objects and their relations to support the DIMMA runtime
- component specifications: define the C++ classes for the engineering objects
- implementation.

4.2 Major components

The overall structure of the EM is shown in Figure 4.1.

4.2.1 Stubs

Stubs implement access transparency. Stubs are generated by a stub compiler. It is still an open issue whether we generate stubs from IDL or directly from the source.

Apart from the normal stubs for operations, signals/streams need more thought. Signals differs from operations in several aspects:

- server operations have upcall dispatches, while signals don't
- operations control the default memory allocation, release for marshalled parameters, while frames for streams may wish to have direct control of memory allocations
- signals require different message queuing and handling support than operations.

If dynamic interface invocation is to be supported, a marshalling interpreter is needed to support dynamic typed invocations.

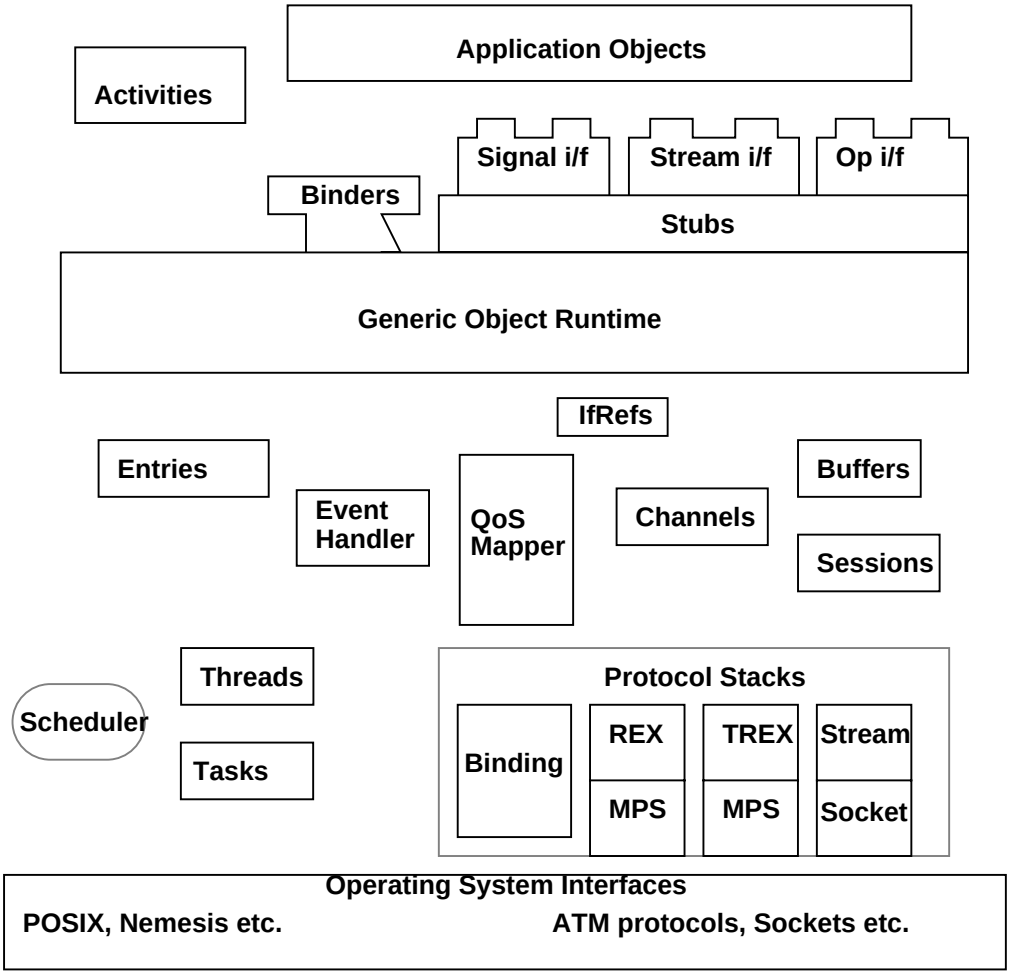
4.2.2 Runtime

The runtime provides an interface for application objects to the engineering Nucleus functions.

4.2.3 Channels

Channels provide engineering endpoints for the interactions between objects.

Figure 4.1: Engineering Model Components



A channel provides an abstraction that hides away the details of the individual communication stacks and their management.

4.2.4 Sessions

Sessions provide the intermediate cache for storing the end-to-end information for each interaction between objects.

4.2.5 Generic communication stack

A generic communication stack should be designed to

- allow the co-existence of multiple protocol stacks
- define the generic interfaces for message processing
- define the generic interfaces for communication management
- provide the generic interfaces for communication endpoints binding
- support the generic interfaces for QoS domains

4.2.6 Execution protocols

Separate execution protocols are used for

- RPC style interactions

- fragmentation
- real-time RPC interactions
- stream transportation
- group interactions

4.2.7 Message passing service

Message passing service (MPS) provide the abstraction hiding away the detailed communication interfaces such as TCP, UDP, IPC etc.

4.2.8 QoS mapper

QoS mapper interprets QoS parameters associated with binding and interface instantiation operations, and invokes the relevant binders to associate QoS with communication endpoints. QoS mapper also provide the generic functions for QoS conformance check and negotiation.

The mapper for invocation QoS needs more thought. It looks too heavy to invoke an QoS interpreter for each invocation.

4.2.9 Implicit binder

A binder relates an interface with a channel endpoint before any interaction can be initiated.

The implicit binder provides the late binding scheme for optimized resource management and is the key mechanism for system scaling.

4.2.10 Explicit binders

The explicit binder provides the early binding scheme for the association of QoS with a channel. Different QoS domain and object interaction scheme requires different explicit binders.

4.2.11 Threads

Threads provide the resources for logical concurrency.

4.2.12 Tasks

Tasks provide the real physical resources for threads to be executed.

4.2.13 Entries

Entries provide scheduling points so that different scheduling concerns can be identified.

4.2.14 Buffers

Buffers are the managed memory units for messages

4.2.15 Scheduler

A scheduler is needed for task/thread allocation and thread synchronization. The scheduler is not needed if kernel threads are used for running tasks.

4.2.16 Events handler

An event handle provides generic service of timing etc.

4.3 Documentation

The EM work will produce the following technical reports:

- DIMMA engineering model
- Component specifications

Other relevant documents are [APM.1207]. [APM.1051] and [APM.101].

References

[LINDEN 93]

van der Linden R. J., *An Overview of ANSA*; **AR.000.00**, APM Ltd., Cambridge U.K., May 1993.