**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**

# ODP C++ API (work in progress presentation)

**Dave Otway**

**Abstract**

.A set of C++ classes and coding rules for developing ODP conforming applications is being prototyped. These will provide portable computational and engineering interfaces to a range of underlying DPEs.

A set of stub generators that generate ODP C++ code from a range of IDLs is being prototyped using the AST tools being developed by workpackage C4a.

APM.1415.01 **Approved** 24th February 1995

Project Management (confidential to ANSA consortium for 2 years)

**Distribution:**
**Supersedes**:
**Superseded by**:


# ODP C++ API

## Work in progress

***Dave Otway***

**djo@ansa.co.uk**

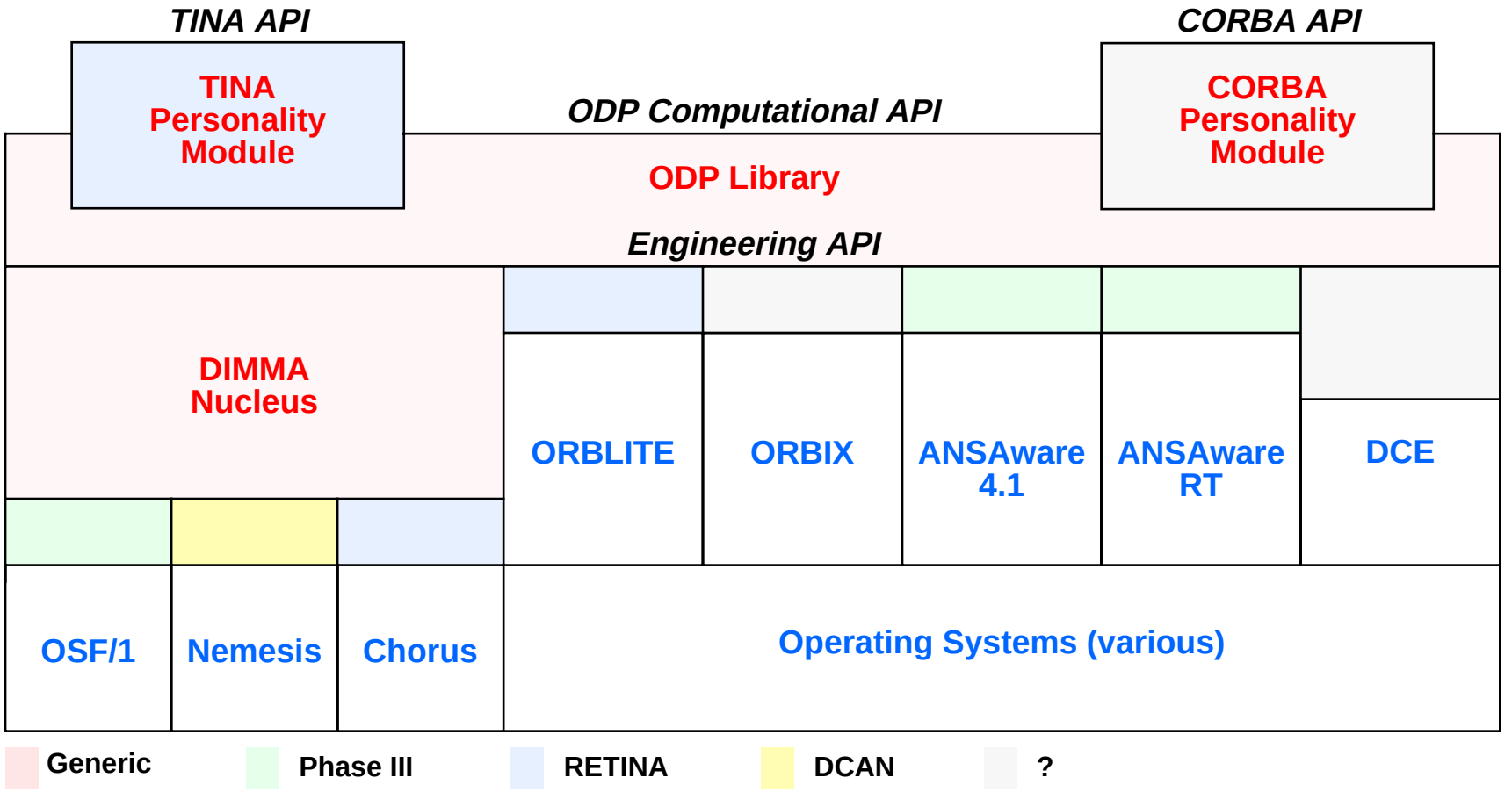

# Background

- ODP Computational model is now an ISO standard
- CORBA is becoming the “standard” DPE product
- We can’t get immediate access to a full function CORBA source
- Phase III requires a prototype DIMMA DPE
- RETINA (and TINA-C) have a requirement for a Telecomms ORB with Real-Time capabilities
- DCAN requires a Real-Time DPE for ATM network management
- C++ is taking over from C as the “standard” implementation language
  - implementations of templates and exceptions are spreading


# Objectives

- pull all the strands together
- build prototype DPE components that can be combined together to satisfy the requirements of all related projects with minimal overlap and reworking
- understand, explain and map the semantic differences between the different APIs, IDLs and DPEs
- design and implement a full function ODP conforming DIMMA DPE
- provide compatibility with CORBA and TINA
  - interoperability by mapping IDLs and protocols
  - application portability by mapping APIs
  - platform portability by adapting to different underlying DPEs

# API & DPE Master Plan




# Computational API

- **Stage 1:**

| | |
|---|---|
| objects | (operational) signatures |
| invocation references | (operational) interfaces |
| multiple results | named terminations |
| basic types | local garbage collection |
| any | (hand coded) trader server stub |
| tracing | capsule manager |

- **stage 2: structured types, threads, service withdrawal**
- **stage 3: explicit binding, QoS, streams, synchronous programming**
- **stage 4: preprocessor**


# Engineering API

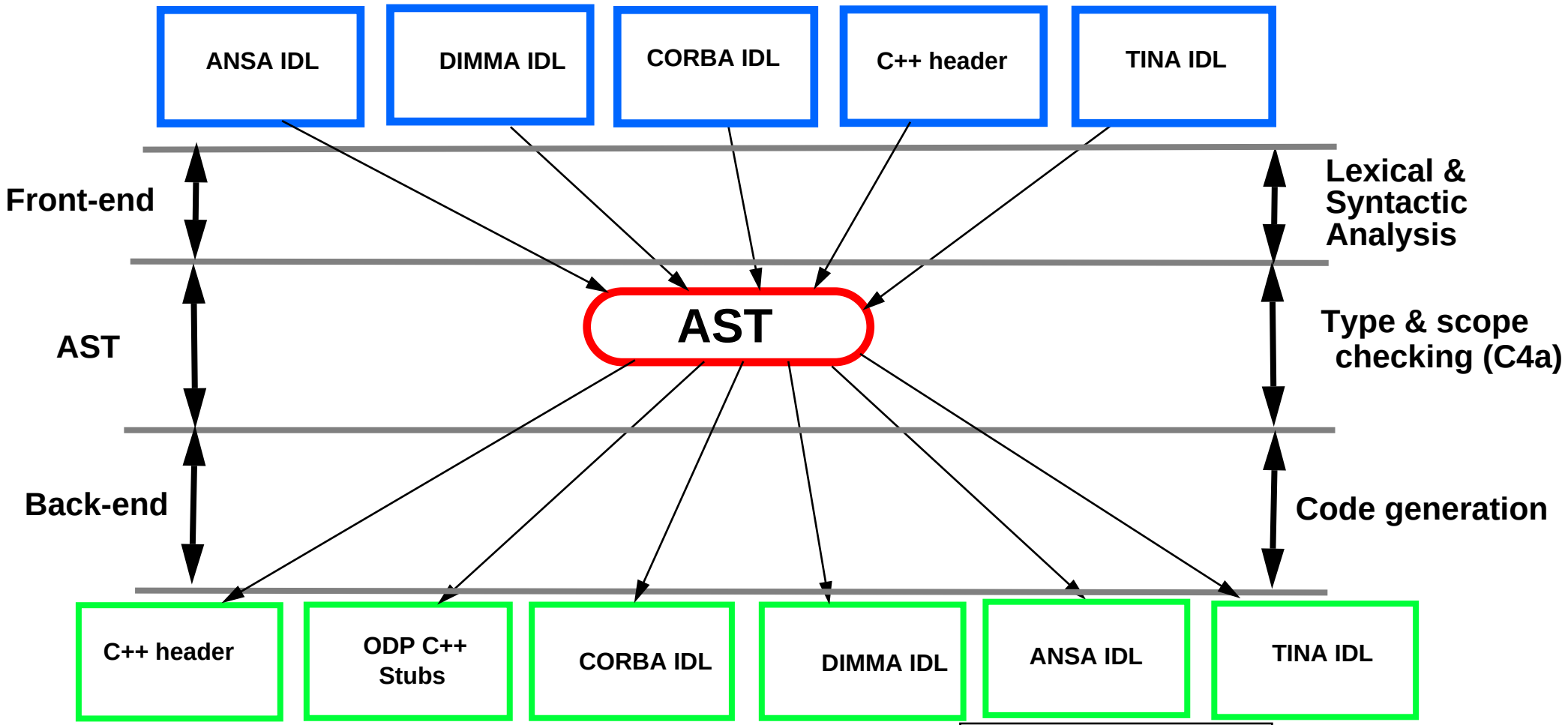
- supports a portable implementation of the computational API and stubs -- the odp library
- on some DPEs, the engineering may lack certain features
  - this must be reported in the earliest epoch possible
- client and server stubs are completely protocol independent
  - base client and server stubs are inherited by the generated stubs
  - buffer (objects) are protocol specific, with their own marshalling/ unmarshalling, sizes and overflow policies
  - the same engineering interface is used for stubs, DII and DSI
- development of the engineering API will track the computational API


# ODP-AST based IDL compiler

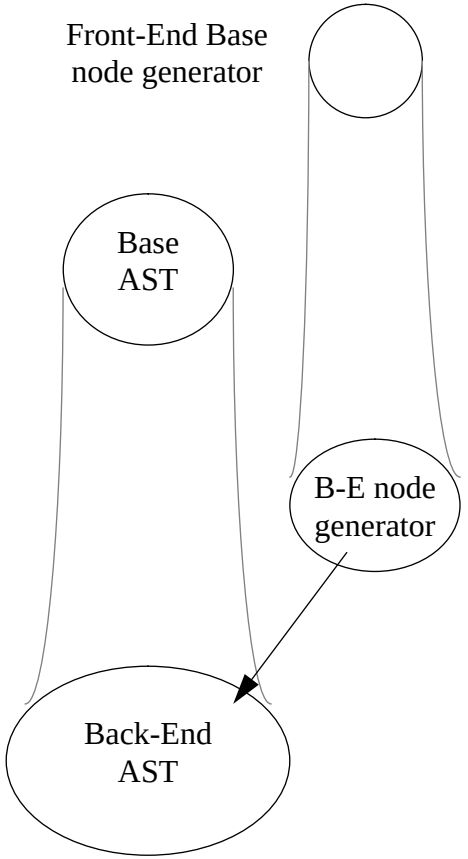
## Objectives:

- support several input notations to describe service interfaces
  - (DIMMA IDL, CORBA IDL, TINA IDL, ANSA IDL, C++ headers)
- generate C++ headers, client and server stubs for the ODP engineering API.
- translate an input IDL notation to another equivalent IDL notation, flagging any incompatibilities
  - (e.g DIMMA IDL -> CORBA IDL)
- reuse the C4a work on conformance based type inferencing

# IDL Master Plan



# AST Design Issues




- back-ends, type checker and front-ends are totally independent from each-other
- base AST nodes capture ODP semantics
  - contain type inferencer & conformance checker
- front-ends create AST via a base node generator
- back-ends specialise the AST nodes for efficent code generation (preserving ODP semantics) and provide a derived node generator
- AST is independent of the IDL grammar and parsing technique
- design enables existing parsers to be plugged in
  - e.g. Sun's public-domain CORBA IDL parser


# Progress

- stage 1 computational and engineering APIs almost complete
  - minus trader stub, capsule manager and type codes
  - the type Any needs tidying up
- a test application has been written and the stubs hand-coded
  - local client and server compile, link and run
  - remote server compiles (but can't link until its got a DPE adaptor)
  - remote client is just missing type codes
- built the DIMMA IDL front-end (lexical, syntactic analysis and AST generation)
- built the infrastructure which will host and link different front-ends and back-ends, and defined the AST interface to both of them
- started coding a back-end for generating C++ stubs and header files


# Plans

- **by May TC**
  - **an implementation of stage 1 computational and engineering APIs running over ANSAware 4.1 (on HP-UX and possibly OSF/1)**
  - **with some stage 2 features (service withdrawal & some structured types)**
  - **an implementation of the C++ stubs and header file back-end**
  - **an adaption of Sun's Public-domain CORBA IDL front-end to our infrastructure**