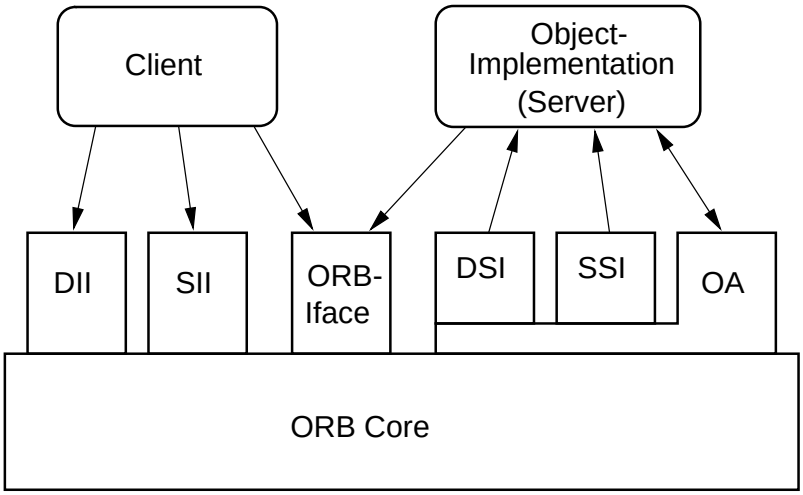
Design and Implementation of the Portable Object Adapter

Frank Pilhofer
Universität Frankfurt
fp@informatik.uni-frankfurt.de

Object Adapters in CORBA

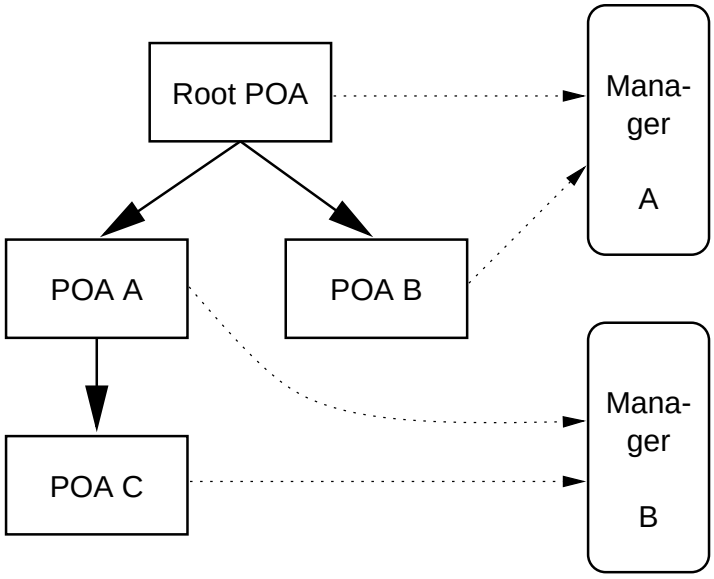


- Activation and deactivation of objects
- Generation of object references
- Synchronization
- Mapping of object references to implementations
- Method invocations

Portable Object Adapter (POA)

- Clear distinction of the terms “object,” “object reference,” and “servant”
- Explicit activation and deactivation of objects
- Support for “virtual” Object References
- Transparent object activation upon first use
- Very dynamic through use of default servants and servant managers

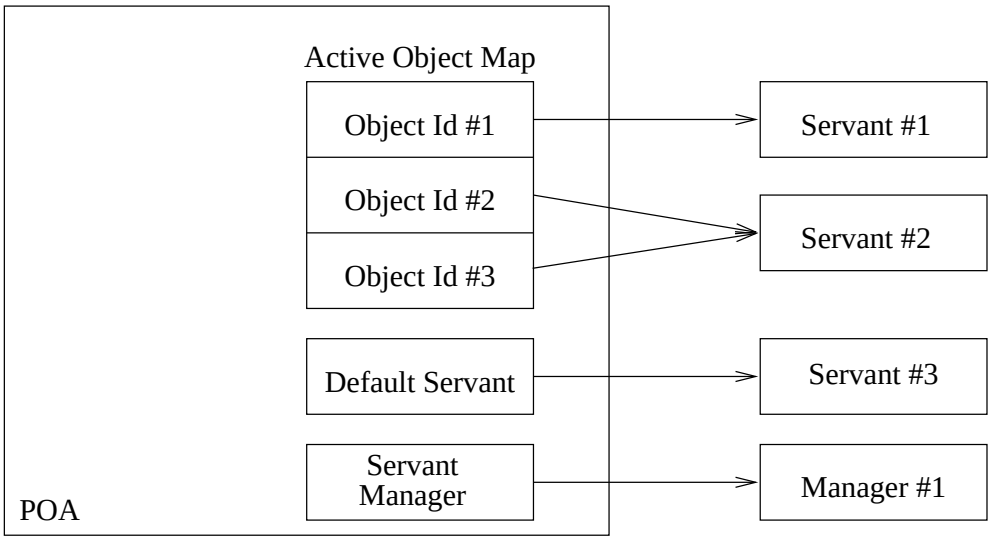
POA Architecture



- Hierarchical structure of many POAs, each with its own set of “policies”

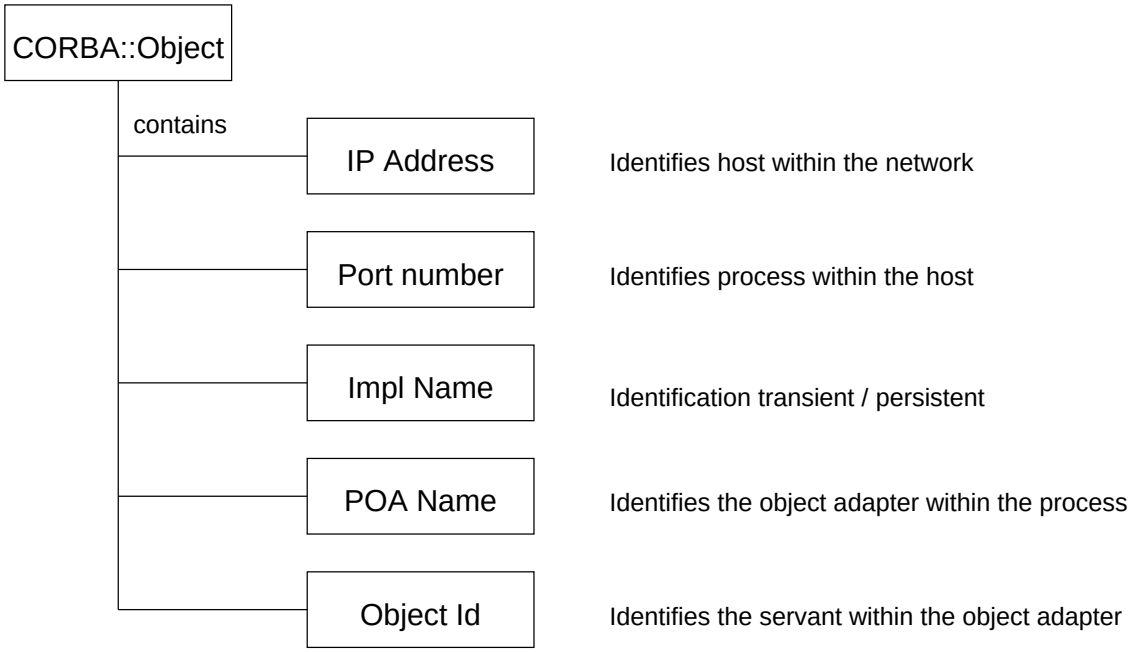
- “POA Managers” for Synchronization

Server-side View of Objects



- Each POA has its own table of active servants
- Each servant is identified by one or many “object ids” that it can serve
- In addition, a “default servant” or a “servant manager” can be registered

Composition of Object References



- Contains all necessary information to identify the responsible servant

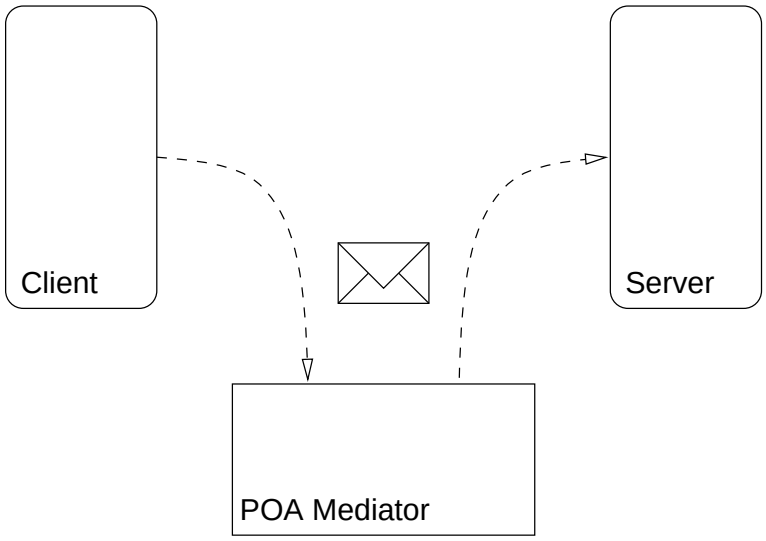
- Only the Root POA is registered with the ORB and dispatches invocations to the responsible POA first

- The responsible POA identifies the servant and performs the invocation

Example Server

```
#include "hello.h"

class Hello_impl : virtual public POA_Hello
{
public: void hello () {printf ("Hello World!\n");}
};

int main (int argc, char *argv[])
{
  CORBA::ORB_var orb =
    CORBA::ORB_init (argc, argv, "mico-local-
orb");

  CORBA::Object_var poaobj =
    orb->resolve_initial_references ("RootPOA");
  PortableServer::POA_var poa =
    PortableServer::POA::_narrow (poaobj);
  PortableServer::POAManager_var mgr =
    poa->the_POAManager();

  Hello_impl * hello = new Hello_impl;

  PortableServer::ObjectId_var oid =
    poa->activate_object (hello);

  CORBA::Object_var objref =
    poa->servant_to_reference (hello);

  mgr->activate ();
  orb->run();
}
```

Persistent POAs



- Persistent POAs require a permanently running daemon
- Method invocations are relayed to the active server
- Registration of persistent servers in the Implementation Repository
- Mediator only replaces address information in object reference, no storage required

MICO and the POA

- POA fulfills its expectations:
  well-specified, powerful, portable
- The POA is an object adapter like any other
- No modifications to the ORB
  were necessary
- Modification of the IDL Compiler and
  Mediator
- approx. 4000 lines of C++ source code
- Shipped with MICO since 2.2.0
- Documentation!