minimumCORBA

Presented By
Shahzad Aslam-Mir
Vertel Corporation
<sam@vertel.com>

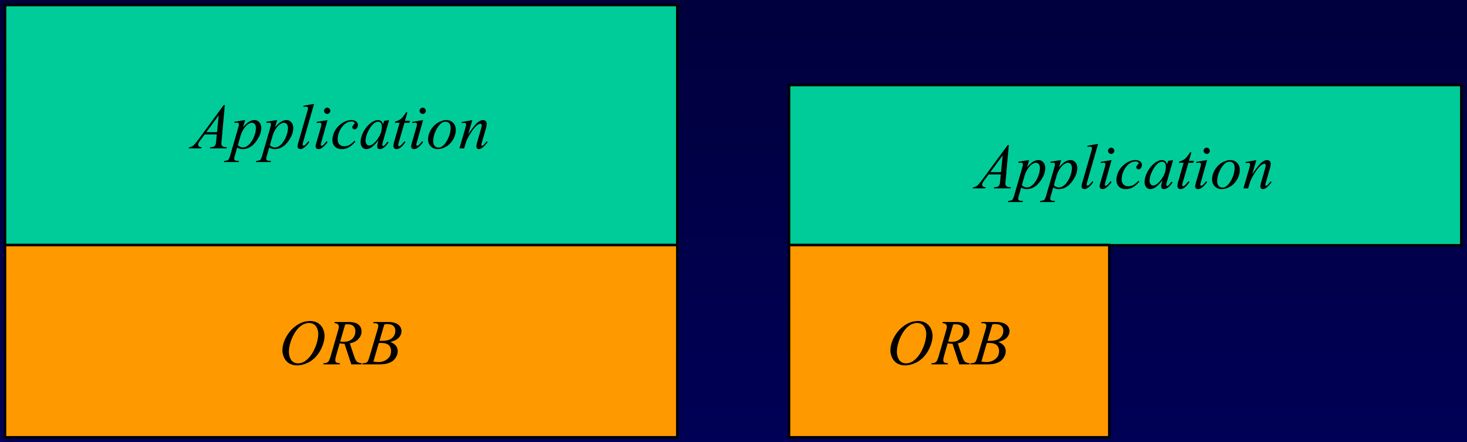
minimumCORBA Philosophy

- A standard profile for limited resource systems
  - Simpler means smaller and faster
  - Vendors can profile implementations … but
    - users need a standard profile
- Fully interoperable with full CORBA
  - Use of minimumCORBA is transparent to external systems
- Portability:
  - Full portability between minimumCORBA systems
  - minimumCORBA portable to full CORBA
  - Full CORBA not necessarily portable to minimumCORBA

minimumCORBA Goal

- Broadly applicable within limited resource systems
- Supports all OMG IDL types for interoperability (even *any*)
- minimumCORBA and “full” CORBA implementations are interoperable
- Save space by eliminating features that support the dynamic aspects of CORBA

Application vs. Platform



- CORBA
  - Rich in features
- Usability
  - High level of abstraction
- High usage of resources

- minimumCORBA
  - Reduced functionality
- Usability
  - Targeted at what's needed
- Uses limited resources

minimumCORBA Profile Overview

- Proper subset of CORBA 2.4
  - nothing has been added
- Dynamic CORBA omitted
  - IR, DII and DSI
  - Static stubs and skeletons only
- ORB, Object and TypeCode interfaces subsetted
- Portable Server framework subsetted
  - some interfaces and some policy options omitted
- Supports full IDL

minimumCORBA Profile - IDL

- Interface Definition Language - full IDL
  - i. e. the language for stubs, not the pseudo objects
  - keep unbounded sequences, anys, inouts
  - maximum interworking with CORBA applications
- Difficulties with full IDL
  - IDL Any type (a dynamic typing concept but its there)
  - context clause on operation signatures (ought to be deprecated but its there)
- Optimizing out what's not used NOT standardized
  - support for unused constructs can be omitted after the fact
    - an application design process issue
    - tool support, e. g. IDL compiler warnings
  - a vendor specific product differentiator

minimumCORBA
IR, DII, & DSI

- Interface Repository
  - omitted as it supports a model where types are discovered at run time rather than known at design time
  - don’t omit RepositoryId formats or pragmas
  - don’t omit TypeCode interface … used for any (next slide)
- Dynamic Invocation Interface
  - omitted as it supports a dynamic programming model where references are only bound to a type at invoke time
- Dynamic Skeleton Interface
  - omitted as it supports a dynamic programming model where Servants are linked to the ORB at run time
- DynAny
  - omitted as it provides an API for anys containing types unknown at compile time

minimumCORBA - TypeCodes

- Need TypeCodes to support IDL any type (full IDL)
- Design time typing assumption
  - a minimumCORBA application cannot handle datatypes undeclared at compile time inside an any
  - when receiving an any, the application can navigate the data with the kind() and the id() - can also use name()
  - retain TypeCode constants to create values of type any for sending
  - no requirement to build up types not pre-declared
  - create_< typecode> operations omitted

minimumCORBA ORB interface

- some signatures omitted
  - create_ list (…), create_ operation_ list (…) - omitted as they support dynamic invocation
  - work_ pending ( ), perform_ work ( ), shutdown (...) – omitted as they support integration with other components (e. g. GUIs) which won't apply to minimumCORBA systems
  - run ( ) retained, for portability reasons
  - get_ default_ context (…)
    - omitted as IDL contexts support an alternate programming style. An identifier- value pair is just an implicitly passed "in string" parameter. This flexibility isn't valuable.
    - The Context object is also omitted
  - get_ current ( ) – omitted as its use is deprecated in CORBA 2.2

minimumCORBA interface Object

- some signatures omitted
  - get_ interface ( ) – omitted as the IR is omitted, interface type issues are decided at design- time
  - get_ implementation ( ) – omitted as its use is deprecated in CORBA 2.2
  - is_ a (…) – omitted as typing issues addressed at design time
  - non_ existent ( )
    - omitted as its not essential for minimumCORBA
    - “Services ... might use this operation in their “idle time” to sift through object tables for objects that no longer exist”.
  - create_ request (…) – omitted: its purpose is form arbitrary requests at run- time

minimumCORBA POA module

- POA { … }
  - omit unecessary policy factory operations
  - omit the_activator attributes (! the_POAManager is retained)
  - omit {get, set}_ servant and {get, set}_ servant_ manager
- Current { … }, Policy { … }
  - objects supported
- POAManager { … }
  - only the activate (…) operation retained
- AdapterActivator { … }, ServantManager { … }
  - omitted as they support a dynamic programming model
  - consequently no ServantActivator or ServantLocator

minimumCORBA POA Policies

- ThreadPolicy = ORB_ CTRL_ MODEL
  - SINGLE_ THREAD_ MODEL omitted, don’t support multi-thread unaware applications on a multi- threaded platform
- ServantRetentionPolicy = RETAIN
  - NON_ RETAIN omitted, it requires other omitted policies
- RequestProcessingPolicy = USE_ ACTIVE_ OBJECT_ MAP_ ONLY
  - USE_ DEFAULT_ SERVANT and USE_ SERVANT_ MANAGER omitted, they support a dynamic model and come at a price
- ObjectIdUniquenessPolicy = UNIQUE_ ID | MULTIPLE_ ID
  - CORBA’s rootPOA has UNIQUE_ ID. MULTIPLE_ ID for FGOs
- IdAssignmentPolicy = SYSTEM_ ID | USER_ ID
  - CORBA’s rootPOA has SYSTEM_ ID. Appln. index as USER_ ID

minimumCORBA Implicit Activation

CORBA 2.4.1, 11.3.3 - “An application server that creates all its needed POAs at the beginning of execution does not need to use or provide an adapter activator; it is necessary only for the case in which POAs need to be created during request processing.”

- Policy = IMPLICIT_ ACTIVATION
  - Servants *may* be activated implicitly
    - of course, Servants can be activated explicitly
- Policy = NO_ IMPLICIT_ ACTIVATION
  - Servants *shall not* be activated implicitly
    - so, Servants must be activated explicitly
- NO_ IMPLICIT_ ACTIVATION is a *subset of* IMPLICIT_ ACTIVATION
  - Suggest, minimum rootPOA has NO_ IMPLICIT_ ACTIVATION

minimumCORBA POA Lifespan Policy

- Requirements from minimumCORBA systems
  - well-known references for rebooting need to be in the system rather than provided by an administrator
  - long-lived clients want transparency for transient server failures, i.e., no need to get a new object reference
  - if client doesn’t invoke during downtime, it need never know
- LifespanPolicy = TRANSIENT
  - CORBA’s rootPOA setting, supports “init & export” model
- LifespanPolicy = PERSISTENT
  - references are the same across epochs, so meets the requirements
  - POAManager:: activate guards against premature invoke

minimumCORBA Miscellaneous

- DCE Interoperability – a separate compliance point, it is omitted
- COM/ CORBA Interworking – a separate compliance point, it is omitted
- Interceptors – omitted as they depend on the Request object (DII) and the ServerRequest object (DSI) which are omitted
- C++ Language mappings
  - no prescriptions: “type unsafe narrow” and “no multiple inheritance” are optimizations that are out of scope
- Java Language mapping
  - Java ORB Portability Interfaces are omitted as they depend on the DII and DSI
  - A subsequent version will provide delegates that don’t rely on DSI