minimumCORBA

Joint Revised Submission

Alcatel
Hewlett-Packard
Inprise Corporation
Iona Technologies, Plc.
Lucent Technologies, Inc.
Northern Telecom
Sun Microsystems, Inc.

supported by:

Highlander Communications, L.C.

OMG TC Document orbos/98-08-04

August 17, 1998

The submitting companies listsed above have all contributed to this “merged” submission. These companies recognize that this draft joint submission is the joint intellectual property of all the submitters, and may be used by any of them in the future, regardless of whether they ultimately participate in a final joint submission.

minimumCORBA Submission 0

0.1 Copyright Waiver

See inside front cover for copyright statement.

0.2 Submission contact points

Zoely Canela
Alcatel Alsthom Recherche
Route de Nozay
91460 Marcoussis
France
phone +33 1 69 63 12 71
fax +33 1 69 63 17 89
E-mail: canela@aar.alcatel-alsthom.fr

Jishnu Mukerji
Hewlett-Packard
300 Campus Drive, MS 2E-62
Florham Park, NJ 07932
phone +1 973 443 7528
fax +1 973 443 7422
E-mail: jis@fpk.hp.com

Jorge Rodriguez
Highlander Communications, L.C.
206 East Pine Street
Lakeland, FL. 33801
phone +1 941 686 7767
E-mail: jorge@highlander.com

Jeff Mischkinsky
Inprise Corporation
951 Mariner's Island Blvd.
San Mateo, CA 94404
phone +1 650 312 5158
fax +1 650 286 2475
E-mail: jeffm@inprise.com

Oisin Hurley
IONA Technologies
8-10 Pembroke St.
Dublin 2
Ireland
phone +353 1 602 2111 x2375
fax +353 1 602 2116
E-mail: ohurley@iona.com

Antonio Rodriguez-Moral
Lucent Technologies
Room 2G-520
101 Crawfords Corner Rd.
Holmdel, NJ 07733
phone +1 732 949 2164
fax +1 732 949 3210
E-mail: arodmor@lucent.com

Dave Stringer
Nortel
Harlow Labs
London Road, Harlow
Essex CM17 9NA, UK
phone +44 1729 403712
fax +44 1279 403930
E-mail: drs@nortel.com

Stephane Carrez
Sun Microsystems
6 av Gustave Eiffel
78182 Montigny Le Bretonneux
France
phone +33 1 30 64 82 22
fax +33 1 30 57 00 66
E-mail: Stephane.Carrez@France.Sun.COM

0.3 Guide to the material in the submission

This chapter provides the justification of, and the supporting text for, the proposed specification. It includes the rationale, the proof of concept and the resolution of RFP requirements and issues to be discussed.

Chapter 1 contains the proposal proper, that is the text for publication as an OMG specification. It comprises: the profile philosophy, the conformance statement followed by subsections detailing the proposed profile of CORBA 2.2 presented along the lines of CORBA 2.2 itself. Finally, the subset of CORBA IDL that constitutes the minimumCORBA IDL is given. All of chapter 1 is normative.

0.4 Rationale

The background of embedded systems tends to require design time decisions on resource allocation, object location and creation. Together with pre-determined patterns of interaction, this yields a much more predictable system environment. As a result the approach adopted in this proposal is to remove the dynamic facilities for creating, activating, passivating and interrogating objects and for serving requests.

Since a minimumCORBA ORB instance will often be part of a larger system containing (full) CORBA ORBs, maximum compatibility and full interoperability has been a design goal.

The term “minimumCORBA” is used to denote the proposed specification in the same manner as “CORBAservices”, “CORBAfacilities” and “CORBAsecurity” denote other parts of the CORBA specification set.

The term “CORBA”, when used to denote a specification, covers the ORB component of the Reference Model (i.e. the core, interoperability and the language mappings for IDL). minimumCORBA defines a profile (or subset) of CORBA, whereas CORBAservices etc. define optional extensions to the CORBA specification.

0.5 Statement of proof of concept

This profile is the result of several years of experience, acquired by the submitting companies, in configuring ORBs for deployment in limited resource scenarios. As such, both the viability and applicability of this profile are proven.

0.6 Resolution of RFP requirements

0.6.1 RFP mandatory requirements

Subset Profiling

This submission defines a single profile of CORBA cut down from CORBA 2.2. While other profiles could be identified, the submitters believe that this is unnecessary and would detract from the overall value of the minimumCORBA concept.

A single profile (or subset) for minimumCORBA is important from both marketing and technical perspectives, as too many profiles will cause confusion. This profile is targeted at a broad range of limited resource systems.

Where applications require a configuration more lightweight than that presented in this submission, then vendors may choose to offer further, link-time flexibility. The ability to

configure a run-time ORB would be a product differentiator.

IDL Profiling

This submission proposes that IDL be retained in full for maximum compatibility with full CORBA.

CORBA Core

This proposal retains the static stubs and skeletons as required. It proposes no extensions to pseudo-IDL for CORBA 2.2 defined objects. In order to maintain maximum compatibility between CORBA and minimumCORBA, the submitters have opted for omission of complete objects where possible.

Effect on Existing CORBA and CORBAservices

Since support for all IDL is retained, there is no bar to accessing any existing CORBA services. Indeed, it is anticipated that instances of minimumCORBA will often run as part of a larger CORBA system and will exploit services run in the full CORBA domain to handle such circumstances as start-up, software download and upgrade. The retention of various services related APIs allows minimumCORBA to be integrated with security and transaction services. It is a vendor option whether to provide these additional capabilities.

0.6.2 RFP optional requirements

In keeping with the "one profile" philosophy, this proposal does not follow the suggested model of optional conformance points. To reiterate, IDL is supported in full, including the any and variable sized datatypes. The core components IR, DII and the DSI are not included in minimumCORBA.

Note that a vendor may choose to supply an IR, for example, along with a minimumCORBA product but would have to describe the offering as "minimumCORBA plus a CORBA conformant IR" as there would still be no full CORBA conformance (e.g. if DII, DSI were omitted).

0.6.3 RFP general requirements

Most of the general requirements described in section 5.1 of the RFP have trivial responses because this proposal is a pure subset and so the response is inherited from statements about CORBA itself. The one exception being the security considerations.

0.6.3.1 Security considerations

Applications based upon minimumCORBA need not provide run-time security. Instead, security may be addressed at design time, for example relying upon the closed nature of the system and physical access policies that will often apply in a minimumCORBA system. A minimumCORBA implementation is at liberty to follow the prescriptions of CORBAsecurity. However, this combination is an extension to minimumCORBA, and

therefore not a conformance point of minimumCORBA.

Applications perform CORBAsecurity related interaction with the ORB via **get_service_information**, via **resolve_initial_references("SecurityCurrent")** and via **get_policy** and **get_domain_managers**. All these operations are present in the minimumCORBA profile. Therefore it is left to vendors whether or not to provide a security capability in their minimumCORBA ORB. The cost of implementing these operations when no security is provided is negligible.

Note that, one possible deployment of components based on minimumCORBA would be constrained within trust domains where the only access to the outside world was via clearly identified interoperability bridges. One way of securing such a system would be to implement an interoperability bridge that additionally provided a minimal security platform.

0.7 Responses to RFP issues to be discussed

The submitters advocate a single conformance point in order to produce an adequate baseline for non-trivial, limited resource applications. The use of link time facilities to selectively include or exclude code on a needs basis is outside the scope of OMG specifications. Vendors may use such techniques, e.g. not requiring code support for "any" types.

The submitters believe that the resultant minimumCORBA ORB is applicable to a significant range of embedded applications. This profile exploits the upfront design activity necessary to create long-lived embedded applications. The profile encourages an implementation paradigm which supports pre-allocation of resources, which in turn results in small fast implementations with all the benefits of distribution as well as access to the full CORBA world.

minimumCORBA Profile 1

This chapter describes minimumCORBA, a subset of CORBA designed for systems with limited resources.

The profile philosophy and a statement of conformance are presented first, followed by a detailed description of the specification, which is presented in sections corresponding to the affected chapters of the CORBA specification.

1.1 Profile Philosophy

For some applications CORBA is too large to meet exacting size and performance requirements. Such scenarios require a cut-down version of CORBA. This cut-down version is called “minimumCORBA”. minimumCORBA defines a profile (or subset) of CORBA, whereas CORBAservices, CORBAsecurity etc. define optional extensions to the CORBA specification.

The features of CORBA omitted by this profile clearly have value in mainstream CORBA applications. However, they are provided at some cost, in terms of resources, and there is a significant class of applications for which that cost cannot be justified.

Features omitted from CORBA could still be implemented by the application in those cases where they are needed. The following figure illustrates the relationship between ORB, application and omitted features..

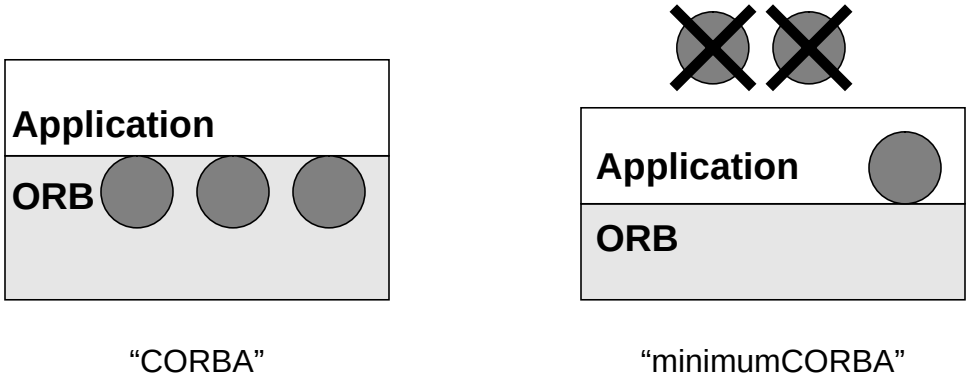

Figure 1-1 Omitting features from CORBA

The omission of a feature of CORBA represents a trade-off between usability and conserving resources: CORBA has a greater degree of user-friendliness whereas minimumCORBA is better for conserving limited resources.

This specification defines a single profile that preserves the key benefits of CORBA: portability of applications and interoperability between ORBs. The following goals are recognized when choosing this profile:

- Which features are retained in minimumCORBA and which are omitted is carefully chosen to yield a profile that still has broad applicability within the world of limited resource systems.
- minimumCORBA should to be fully interoperable with CORBA as applications running on minimumCORBA ORBs may be part of systems that includes components running on CORBA ORBs.
- minimumCORBA should support full IDL. So that, given sufficient resources, any CORBA application can be executed on either full CORBA or on minimumCORBA or partitioned between the two.
- Features that support the dynamic aspects of CORBA are omitted, as the systems for which minimumCORBA is targeted will make design-time commitments, e.g. with regard to interface type checking.

It will always be possible to envisage more constrained environments and so there has to be criteria to determine when the subset is small enough, without sacrificing broad applicability. The line is drawn by referring back to the “portability”, “interoperability” and “full IDL” goals.

Included within the minimumCORBA profile are several features that incur cost, in terms of static ORB size and stub code size, even when the application makes no use of them.

- TypeCode Features: Savings could be made by not supporting type safety with respect to “any”, to TypeCodes and to narrowing of Object References.
- Exception Features: support for both user and system exceptions could be omitted when user exceptions are not used in the application. The reduced programming model would still be useful, e.g. in cooperating finite state machines where objects would “fail safe” and recovery would be handled by the application.
- Inheritance Features: the tables needed to implement the provision of multiple inheritance could be omitted if the application undertakes not to use any multiple IDL inheritance.

Conformant implementations of minimumCORBA may choose to include these optimizations where it can be ascertained that the application does not use them. However, the definition of compiler/linker options is beyond the scope of CORBA specifications. Therefore, these optimizations are not included in the minimumCORBA profile.

1.2 Conformance

This specification defines a single conformance point. This is a proper subset of the CORBA specification. This specification makes no extensions to the specification being subsetted.

minimumCORBA is based upon CORBA 2.2 (OMG document formal/98-02-01).

The profile is defined in terms of explicit omission of specific parts of the specification, so that any part of the CORBA 2.2 specification (as defined in document formal/98-02-01) not explicitly omitted is still part of the minimumCORBA specification.

Omitted features are not present in the minimumCORBA copy of **module CORBA** and **module PortableServer**, so that the APIs and their semantics are not required to be provided in a conformant implementation.

This specification does not discuss the CORBAservices as they are separate and distinct conformance points for a CORBA implementation.

1.3 IDL

minimumCORBA supports all of OMG IDL, as defined in chapter 3 of the CORBA specification. This allows maximum compatibility between minimumCORBA and full CORBA applications.

1.4 ORB Interface

A number of omissions are made from the ORB interface, as defined in chapter 4 of the CORBA specification:

1.4.1 ORB

The **create_list** and **create_operation_list** operations are omitted, as their purpose is to support the DII.

The **work_pending**, **perform_work** and **shutdown** operations are omitted, as they are only needed for certain styles of CORBA application, and are not required for basic ORB operation. Note that the **run** operation is retained as it is important, in a single threaded model, to provide the server initialization code with a portable entry point to the ORB. In a multi-threaded model, **run** can be implemented as a wrapper for the appropriate threading primitive.

The Context object is omitted as it is defined as part of the DII and only adds support for an alternate programming style : using identifiers in a **context** clause differs from using additional **in string** arguments only in that the former are passed implicitly, whereas the latter have to be provided as actual parameters in the function call. As the Context object is omitted, the **get_default_context** operation is omitted.

Note that the **context** keyword is still present in minimumCORBA IDL. However, due to the omission of the Context Object, there is no standard interface for a client to associate values with context identifiers. Where an IDL signature defines a **context** but no values are available at the time of invocation, IIOP requires an empty sequence to be passed. On the server side, a minimumCORBA application could not retrieve the values associated with context identifiers by a client CORBA application. Interoperability is maintained at a syntactic level only.

The **get_current** operation is omitted from minimumCORBA, as it is deprecated from CORBA 2.2.

1.4.2 Object

The **get_interface** operation is omitted from minimumCORBA, as the Interface Repository is omitted.

The **get_implementation** operation is omitted, as it is deprecated in CORBA 2.2.

The **is_a** operation is omitted in order not to introduce a requirement either for holding detailed type information in the object reference or for getting type information over the wire. Instead, minimumCORBA relies on design time resolution of type checking issues.

The **non_existent** operation is omitted, because of the design philosophy of making more decisions statically, at design time.

The **create_request** operation is omitted, as the Dynamic Invocation Interface is omitted.

1.4.3 ConstructionPolicy

The ConstructionPolicy interface and its supporting constant, SecConstruction, are omitted. It is not necessary for minimumCORBA applications to organize their constituent objects into different policy management domains. Consequently all minimumCORBA objects will belong to the default domain for the ORB and so, if there is no default, belong to no domain.

1.5 Dynamic Invocation Interface

The entire Dynamic Invocation Interface, as defined in chapter 5 of the CORBA 2.2 specification is omitted from minimumCORBA. Note that this means that the NamedValue type and NVList are omitted too.

1.6 Dynamic Skeleton Interface

The entire Dynamic Skeleton Interface, as defined in chapter 6 of the CORBA 2.2 specification, is omitted from minimumCORBA.

1.7 Dynamic Any

Dynamic Anys, as defined in chapter 7 of the CORBA 2.2 specification, are omitted from minimumCORBA.

1.8 Interface Repository

The majority of the Interface Repository, as defined in chapter 8 of the CORBA 2.2 specification, is omitted from minimumCORBA, as it is part of the dynamically typed programming model. There are two exceptions: the RepositoryIds, for which formats and pragmas are defined in section 8.6; and the TypeCode interface, as defined in section 8.7, for which a minimumCORBA version is retained.

The pragmas enable type id information to be changed, which can, among other things, be used to implement a more compact type naming convention. The pragmas may be acted upon or ignored by an implementation of minimumCORBA, as this is the same semantics as the CORBA 2.2 specification.

The TypeCode interface is included because of its role in the semantics of the **any** type. when using the CORBA **any** type, an application in a minimumCORBA domain will only send and receive IDL types that were known at build time. Hence, part of the TypeCode interface is omitted.

1.8.1 TypeCode

The **id**, **kind** and **name** operations are retained. They are sufficient to allow applications to distinguish types known at build time. Other operations that support arbitrary constructed and template types are omitted as a minimumCORBA application is not expected to handle these arbitrary types. The operations omitted are: **member_count**, **member_name**, **member_type**, **member_label**, **discriminator_type**, **default_index**, **length**, **content_type**, **fixed_digits**, **fixed_scale**, **param_count** and **parameter**. The **Bounds** exception is also omitted as it is only used by omitted operations.

All the TypeCode create operations are omitted from the ORB interface as they support the creation of **any** values that have types created dynamically. In a minimumCORBA application, TypeCodes are created as constants by the programmer or by tools (e.g. an IDL compiler). The operations omitted are: **create_struct_tc**, **create_union_tc**, **create_enum_tc**, **create_alias_tc**, **create_exception_tc**, **create_interface_tc**,

create_string_tc, **create_wstring_tc**, **create_sequence_tc**, **create_recursive_sequence_tc** and **create_array_tc**.

1.9 Portable Object Adaptor

minimumCORBA supports a subset of the interfaces and policies defined in chapter 9 of the CORBA 2.2 specification. The interfaces and policies that are not supported are omitted from the minimumCORBA copy of **module PortableServer**.

1.9.1 Interfaces

POA

The POA object is profiled in minimumCORBA with items omitted where they support a dynamic mode of POA operation. What remains is sufficient to achieve portability and interoperability between different minimumCORBA implementations and between minimumCORBA and full CORBA.

The following policy object factory operations are omitted: **create_thread_policy**, **create_implicit_activation_policy**, **create_servant_retention_policy** and **create_request_processing_policy**.Only the default values for the associated policies are supported and so there is no requirement to create these policy objects.

The **the_activator** attribute is omitted as minimumCORBA does not support dynamic (on demand) activation of POAs.

The **get_servant_manager** and **set_servant_manager** operations are omitted as minimumCORBA omits ServantManagers.

The **get_servant** and **set_servant** operations are omitted as minimumCORBA doesn't support the USE_DEFAULT_SERVANT option for the RequestProcessingPolicy.

Current

The PortableServer::Current object is fully supported, again for reasons of portability and interoperability.

Policy interfaces

The Policy objects and their associated policy value enums are omitted where the only supported value is the default value as in these cases there is no requirement to the policy objects. Where more than one policy value is supported the policy object and associated enum remains. This is sufficient to support portability and interoperability. The policy objects omitted are: ThreadPolicy, ImplicitActivationPolicy, ServantRetentionPolicy and RequestProcessingPolicy. Also see section 1.9.2.

POAManager

The POAManager object remains in minimumCORBA as the type is used in the

create_POA operation. The only declarations not omitted are the **activate** operation and the **AdaptorInactive** exception. The other declarations in the POAManager interface are omitted from minimumCORBA, as they add extra functionality not required for basic ORB operation. The **activate** operation is retained as it provides portability of minimumCORBA applications to CORBA environments.

AdapterActivator

The AdapterActivator object is omitted from minimumCORBA, because it supports a dynamic mode of POA operation that is not required for basic ORB operation.

ServantManagers

The ServantManager object is omitted from minimumCORBA. This is because it supports a dynamic mode of operation that is not required for basic ORB operation. Consequently, both the derived interfaces, ServantActivator and ServantLocator, are omitted. The PortableServer::ForwardRequest exception is also omitted as it can only be raised by operations of the omitted, derived interfaces.

1.9.2 Policies

The policies supported include all of the default policy values from CORBA. The minimumCORBA RootPOA is a subset of the CORBA RootPOA. The only policy in which it differs is more restrictive than its CORBA RootPOA counterpart. Hence an application built on the minimumCORBA RootPOA will run on the CORBA RootPOA.

ThreadPolicy

The only minimumCORBA ThreadPolicy is ORB_CTRL_MODEL. The SINGLE_THREAD_MODEL policy is omitted because it is not required for basic ORB operation.

LifespanPolicy

minimumCORBA supports both values of LifespanPolicy - TRANSIENT and PERSISTENT. The PERSISTENT policy is retained because it allows the creation of ‘well known’ object references, which allow a service to still be contacted using the same reference after it has been re-initialized. This is useful in a constrained resource environment, as it allows applications to dispense with code to re-obtain references for servers.

Note that minimumCORBA takes the PERSISTENT policy to imply nothing more than the converse of the TRANSIENT policy. That is, that using the PERSISTENT policy, object references generated using one instantiation of a POA may be successfully used after the POA is deactivated and reinstantiated in another process. No further action to restore the state of the POA or the objects managed by it is assumed.

As minimumCORBA does not support Adapter Activators or Servant Managers, minimumCORBA applications implementing a POA with the PERSISTENT policy are

responsible for re-creating the POA and re-activating the relevant objects before these objects can be successfully invoked upon from clients still holding references to them from previous instantiations of the POA.

ObjectIdUniquenessPolicy

minimumCORBA supports both values of ObjectIdUniquenessPolicy - UNIQUE_ID and MULTIPLE_ID - as the cost of the latter is negligible and it offers the ability to save resources by multiplexing multiple objects onto one servant.

IdAssignmentPolicy

minimumCORBA supports both values of IdAssignmentPolicy: SYSTEM_ID and USER_ID. The cost of having both is negligible and is useful in a constrained resource environment, as it allows the re-use in ObjectIds of values that have a meaning in another context within an application.

ServantRetentionPolicy

minimumCORBA only supports the RETAIN ServantRetentionPolicy. The NON_RETAIN policy is omitted in accordance with the design policy of removing dynamic behaviours which are not necessary to basic operation. The dynamic model it supports has non-negligible cost and implications for system predictability.

RequestProcessingPolicy

minimumCORBA only supports the USE_ACTIVE_OBJECT_MAP_ONLY RequestProcessingPolicy. The USE_DEFAULT_SERVANT and USE_SERVANT_MANAGER policies are omitted for the same reasons as the NON_RETAIN option.

ImplicitActivationPolicy

minimumCORBA supports only the NO_IMPLICIT_ACTIVATION policy. IMPLICIT_ACTIVATION is omitted as it is not required for basic ORB operation, and the dynamic programming model it supports has non-negligible cost.

For this policy, minimumCORBA is aligned with the default policy value in CORBA. The CORBA RootPOA has an ImplicitActivationPolicy of IMPLICIT_ACTIVATION. However, the minimumCORBA RootPOA is still a subset of the CORBA RootPOA because the IMPLICIT_ACTIVATION setting does not prohibit explicit activation and the NO_IMPLICIT_ACTIVATION setting permits only explicit activation. That is the one permitted activation mode in minimumCORBA is one of the two permitted activation modes of CORBA.

1.10 Interoperability

minimumCORBA has the same conformance criteria regarding interoperability as

CORBA 2.2, as described in chapters 10 to 13 of the CORBA 2.2 specification. The positioning of interoperability conformance with respect to the CORBA APIs is illustrated in the following figure.

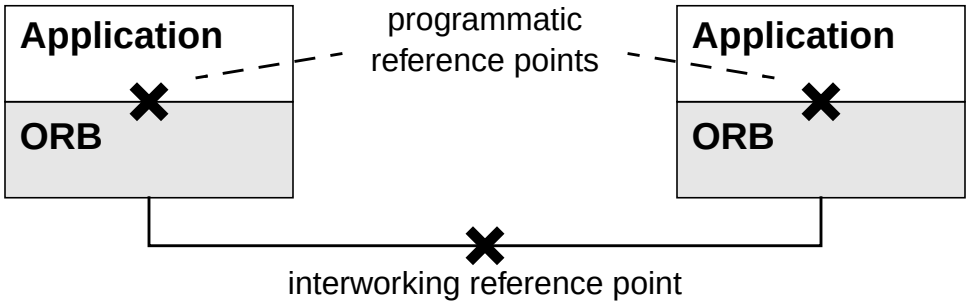

Figure 1-2 Reference points for CORBA conformance

The key thing to notice in this picture is that the interworking reference point, where CORBA interoperability is defined, is different in nature to the programmatic reference points. The former is a protocol whilst the latter are the client and server side APIs. The CORBA specification makes only a limited coupling between the two. For example, the **is_a** API need not result in a "_is_a" request message.

1.11 DCE interoperability

The DCE ESIOP, as defined in chapter 14 of the CORBA 2.2 specification, is omitted from minimumCORBA.

1.12 COM/CORBA interworking

Interworking between COM and CORBA, as defined in chapters 15, 16 and 17 of the CORBA 2.2 specification, is omitted from minimumCORBA.

1.13 Interceptors

Interceptors, as defined in chapter 18 of the CORBA 2.2 specification, are omitted from minimumCORBA, as they depend on the DII and DSI.

1.14 Language Mappings

minimumCORBA implementations must support at least one language mapping as defined by the OMG. However, no specific language binding is mandated.

For each supported language binding, the full mapping must be supported except for those core objects that have been omitted. In the case of the C++ and Java mappings there are further omissions described below.

1.14.1 C++ Mapping Specific Issues

All of the C++ mapping is retained in minimumCORBA except for those elements that result from omitted features of **module CORBA** and **module PortableServer**.

A further omission concerns the semantics of the **`_this()`** member function. It is not possible for **`_this()`** to cause implicit activation of the servant in a minimumCORBA application.

As noted in section 1.1, conformant minimumCORBA ORB implementations may offer optimizations that optionally remove code required for the support of features such as type-safe narrowing and multiple inheritance of IDL interfaces, that incur code size cost even when they are not used. However, these optimizations are vendor specific enhancements, and are not included in the minimumCORBA profile.

1.14.2 Java Mapping Specific Issues

All of the Java mapping is retained in minimumCORBA except for those elements that result from omitted features of **module CORBA** and **module PortableServer**.

One further omission concerns the Java ORB Portability Interfaces, as defined in section 24.18 of CORBA 2.2, which are also omitted from minimumCORBA. This is because they depend on the DII and DSI, which are omitted from minimumCORBA.

A subsequent version of CORBA is expected to provide static portable Java stubs. Once they are specified it will be possible to update the minimumCORBA profile to include them.

1.15 minimumCORBA IDL

The following sub-sections detail the minimumCORBA subset of CORBA IDL. Each section corresponds to a chapter of the CORBA specification, and indicates what part, if any, of the IDL in that chapter is included in minimumCORBA IDL.

Where all or part of the IDL in a chpater of the CORBA specification is included in minimumCORBA, the full IDL from CORBA is shown, with those parts that are omitted from minimumCORBA struck through.

Where all of the IDL in a chapter of the CORBA specification is omitted from minimumCORBA, this is just stated, rather than listing the IDL with every line struck through.

The minimumCORBA **module CORBA** and its counterpart in CORBA 2.2 are distinguished by their contents and not by an IDL identifier or version indicator. The need to distinguish two modules cannot be met by varying the name (i.e. CORBA) or by varying #pragma prefix (i.e. omg.org) or #pragma version (i.e. 2.2), even if the CORBA 2.2 modules contained #pragmas, because this would lead to different fully scoped names and repository ids. That in turn would compromise portability and interoperability. Note the same is true for **module PortableServer**.

Instead it is left to vendors to address the usability concerns in a manner appropriate to their product. For example, toolsets could include a switch for minimumCORBA mode or IDL compilers could include files from different paths. As toolsets and compilers are beyond the scope of CORBA specifications, neither of these possibilities are prescribed.

1.15.1 ORB Interface

```
module CORBA {
 typedef unsigned short ServiceType;
 typedef unsigned long ServiceOption;
 typedef unsigned long ServiceDetailType;

 const ServiceType Security = 1;

 struct ServiceDetail {
  ServiceDetailType  service_detail_type;
  sequence <octet>  service_detail;
 };

 struct ServiceInformation {
  sequence <ServiceOption>  service_options;
  sequence <ServiceDetail>  service_details;
 };

 interface  ORB {
  string object_to_string (in Object obj);
  Object   string_to_object (in string str);

  Status   create_list (
   in long   count,
   out NVList   new_list
  );

  Status   create_operation_list (
   in OperationDef   oper,
   out NVList   new_list
  );

  Status get_default_context (out Context ctx);

  boolean get_service_information (
   in ServiceType  service_type;
   out ServiceInformation  service_information;
  );

  // get_current deprecated operation - should not be used by new code
  // new code should use resolve_initial_reference operation instead
  Current get_current();
```

```
  //Obtaining Initial Object References

    typedef string ObjectId;
    typedef sequence <ObjectId> ObjectIdList;

    exception InvalidName {};

  ObjectIdList list_initial_services ();

  Object resolve_initial_references (in ObjectId identifier)
     raises (InvalidName);

  boolean work_pending();
  void perform_work();
  void shutdown( in boolean wait_for_completion );
  void run();

};


interface Object {          // PIDL
  ImplementationDef get_implementation ();
  InterfaceDef get_interface ();
  boolean  is_nil();
  Object   duplicate ();
  void   release ();
  boolean is_a (in string logical_type_id);
  boolean non_existent();
  boolean is_equivalent (in Object other_object);
  unsigned  long hash(in unsigned long maximum);

  Status create_request (
    in Context  ctx,
    in Identifier operation,
    in NVList arg_list,
    inout NamedValueresult,
    out Request request,
    in Flags    req_flags
  );

  Policy get_policy (
    in PolicyType policy_type
  );

  DomainManagersList get_domain_managers ();

};


//ORB Initialization
typedef string ORBid;
```

```
typedef sequence <string> arg_list;
ORB ORB_init (inout arg_list argv, in ORBid orb_identifier);

//Current Object
interface Current {
};

//Policy Object
typedef unsigned long PolicyType;

// Basic IDL definition
interface Policy {
  readonly attribute PolicyType policy_type;
  Policy copy();
  void destroy();
};

typedef sequence <Policy> PolicyList;


//Domain management operations
interface DomainManager {
  Policy get_domain_policy (
    in PolicyType policy_type
  );
};

const PolicyType SecConstruction = 11;

interface ConstructionPolicy: Policy  {
  void make_domain_manager(
    in CORBA::InterfaceDef object_type,
    in boolean constr_policy
  );

};

typedef sequence <DomainManager> DomainManagerList;

};
```

1.15.2 Dynamic Invocation Interface

As the DII is omitted from minimumCORBA, all of the CORBA IDL for the DII, as defined in chapter 5 of the CORBA specification, is omitted from minimumCORBA IDL.

1.15.3 Dynamic Skeleton Interface

As the DSI is omitted from minimumCORBA, all of the CORBA IDL for the DSI, as defined in chapter 6 of the CORBA specification, is omitted from minimumCORBA IDL.

1.15.4 Dynamic Management of Any Values

As Dynamic Anys are omitted from minimumCORBA, all of the CORBA IDL for Dynamic Anys, as defined in chapter 7 of the CORBA specification, is omitted from minimumCORBA IDL.

1.15.5 Interface Repository

```
module CORBA {
 typedef string Identifier;
 typedef string ScopedName;
 typedef string RepositoryId;

 enum DefinitionKind {
  dk_none, dk_all,
  dk_Attribute, dk_Constant, dk_Exception, dk_Interface,
  dk_Module, dk_Operation, dk_Typedef,
  dk_Alias, dk_Struct, dk_Union, dk_Enum,
  dk_Primitive, dk_String, dk_Sequence, dk_Array,
  dk_Repository,
  dk_Wstring, dk_Fixed
 };

 interface IRObject {
  // read interface
  readonly attribute DefinitionKind def_kind;
  // write interface
  void destroy ();
 };

 typedef string VersionSpec;

 interface Contained;
 interface Repository;
 interface Container;

 interface Contained : IRObject {
  ...
  // Interface contents not shown for brevity
  ...
 };

 interface ModuleDef;
 interface ConstantDef;
 interface IDLType;
 interface StructDef;
 interface UnionDef;
 interface EnumDef;
 interface AliasDef;
 interface InterfaceDef;
 typedef sequence <InterfaceDef> InterfaceDefSeq;

 typedef sequence <Contained> ContainedSeq;

 struct StructMember {
  Identifier name;
  TypeCode type;
```

```
  IDLType type_def;
};

typedef sequence <StructMember> StructMemberSeq;

struct UnionMember {
  Identifier name;
  any label;
  TypeCode type;
  IDLType type_def;
};

typedef sequence <UnionMember> UnionMemberSeq;
typedef sequence <Identifier> EnumMemberSeq;

interface Container : IRObject {
  ...
  // Interface contents not shown for brevity
  ...
};


interface IDLType : IRObject {
  readonly attribute TypeCode type;
};


interface PrimitiveDef;
interface StringDef;
interface SequenceDef;
interface ArrayDef;

enum PrimitiveKind {
  pk_null, pk_void, pk_short, pk_long, pk_ushort, pk_ulong,
  pk_float, pk_double, pk_boolean, pk_char, pk_octet,
  pk_any, pk_TypeCode, pk_Principal, pk_string, pk_objref,
  pk_longlong, pk_ulonglong, pk_longdouble, pk_wchar, pk_wstring
};

interface Repository : Container {
  ...
  // Interface contents not shown for brevity
  ...
};


interface ModuleDef : Container, Contained {
};

struct ModuleDescription {
  Identifier name;
```

```
  RepositoryId id;
  RepositoryId defined_in;
  VersionSpec version;
};


interface ConstantDef : Contained {
  readonly attribute TypeCode type;
  attribute IDLType type_def;
  attribute any value;
};

struct ConstantDescription {
  Identifier name;
  RepositoryId id;
  RepositoryId defined_in;
  VersionSpec version;
  TypeCode type;
  any value;
};


interface TypedefDef : Contained, IDLType {
};

struct TypeDescription {
  Identifier name;
  RepositoryId id;
  RepositoryId defined_in;
  VersionSpec version;
  TypeCode type;
};


interface StructDef : TypedefDef, Container {
  attribute StructMemberSeq members;
};

interface UnionDef : TypedefDef, Container {
  readonly attribute TypeCode discriminator_type;
  attribute IDLType discriminator_type_def;
  attribute UnionMemberSeq members;
};


interface EnumDef : TypedefDef {
  attribute EnumMemberSeq members;
};


interface AliasDef : TypedefDef {
```

```
  attribute IDLType original_type_def;
};

interface PrimitiveDef: IDLType {
  readonly attribute PrimitiveKind kind;
};


interface StringDef : IDLType {
  attribute unsigned long bound;
};

interface WstringDef : IDLType {
  attribute unsigned long bound;
};

interface FixedDef : IDLType {
  attribute unsigned short digits;
  attribute short scale;
};

interface SequenceDef : IDLType {
  attribute unsigned long bound;
  readonly attribute TypeCode element_type;
  attribute IDLType element_type_def;
};


interface ArrayDef : IDLType {
  attribute unsigned long length;
  readonly attribute TypeCode element_type;
  attribute IDLType element_type_def;
};


interface ExceptionDef : Contained, Container {
  readonly attribute TypeCode type;
  attribute StructMemberSeq members;
};

struct ExceptionDescription {
Identifier name;
  RepositoryId id;
  RepositoryId defined_in;
  VersionSpec version;
  TypeCode type;
};
enum AttributeMode {ATTR_NORMAL, ATTR_READONLY};

interface AttributeDef : Contained {
  readonly attribute TypeCode type;
```

```
  attribute IDLType type_def;
  attribute AttributeMode mode;
};

struct AttributeDescription {
  Identifier name;
  RepositoryId id;
  RepositoryId defined_in;
  VersionSpec version;
  TypeCode type;
  AttributeMode mode;
};


enum OperationMode {OP_NORMAL, OP_ONEWAY};

enum ParameterMode {PARAM_IN, PARAM_OUT, PARAM_INOUT};
struct ParameterDescription {
  Identifier name;
  TypeCode type;
  IDLType type_def;
  ParameterMode mode;
};
typedef sequence <ParameterDescription> ParDescriptionSeq;

typedef Identifier ContextIdentifier;
typedef sequence <ContextIdentifier> ContextIdSeq;

typedef sequence <ExceptionDef> ExceptionDefSeq;
typedef sequence <ExceptionDescription> ExcDescriptionSeq;

interface OperationDef : Contained {
  ...
  // Interface contents not shown for brevity
  ...
};

struct OperationDescription {
  Identifier name;
  RepositoryId id;
  RepositoryId defined_in;
  VersionSpec version;
  TypeCode result;
  OperationMode mode;
  ContextIdSeq contexts;
  ParDescriptionSeq parameters;
  ExcDescriptionSeq exceptions;
};


typedef sequence <RepositoryId> RepositoryIdSeq;
```

```
typedef sequence <OperationDescription> OpDescriptionSeq;
typedef sequence <AttributeDescription> AttrDescriptionSeq;

interface InterfaceDef : Container, Contained, IDLType {
  ...
  // Interface contents not shown for brevity
  ...
};


enum TCKind {
  tk_null, tk_void,
  tk_short, tk_long, tk_ushort, tk_ulong,
  tk_float, tk_double, tk_boolean, tk_char,
  tk_octet, tk_any, tk_TypeCode, tk_Principal, tk_objref,
  tk_struct, tk_union, tk_enum, tk_string,
  tk_sequence, tk_array, tk_alias, tk_except
  tk_longlong, tk_ulonglong, tk_longdouble,
  tk_wchar, tk_wstring, tk_fixed
};

interface TypeCode { // PIDL
  exception Bounds {};
  exception BadKind {};

  // for all TypeCode kinds
  boolean equal (in TypeCode tc);
  TCKind kind ();

  // for tk_objref, tk_struct, tk_union, tk_enum, tk_alias, and tk_except
  RepositoryId id () raises (BadKind);

  // for tk_objref, tk_struct, tk_union, tk_enum, tk_alias, and tk_except
  Identifier name () raises (BadKind);

  // for tk_struct, tk_union, tk_enum, and tk_except
  unsigned long member_count () raises (BadKind);
  Identifier member_name (in unsigned long index) raises (BadKind,
    Bounds);

  // for tk_struct, tk_union, and tk_except
  TypeCode member_type (in unsigned long index) raises (BadKind,
    Bounds);

  // for tk_union
  any member_label (in unsigned long index) raises (BadKind, Bounds);
  TypeCode discriminator_type () raises (BadKind);
  long default_index () raises (BadKind);

  // for tk_string, tk_sequence, and tk_array
  unsigned long length () raises (BadKind);
```

```
  // for tk_sequence, tk_array, and tk_alias
  TypeCode content_type () raises (BadKind);

  // for tk_fixed
  unsigned short fixed_digits() raises (BadKind);
  short fixed_scale() raises (BadKind);

  // deprecated interface
  long param_count ();
  any parameter (in long index) raises (Bounds);
};

interface ORB {
  // other operations ...

  TypeCode create_struct_tc (
    in RepositoryId id,
    in Identifier name,
    in StructMemberSeq members
  );

  TypeCode create_union_tc (
    in RepositoryId id,
    in Identifier name,
    in TypeCode discriminator_type,
    in UnionMemberSeq members
  );

  TypeCode create_enum_tc (
    in RepositoryId id,
    in Identifier name,
    in EnumMemberSeq members
  );

  TypeCode create_alias_tc (
    in RepositoryId id,
    in Identifier name,
    in TypeCode original_type
  );

  TypeCode create_exception_tc (
    in RepositoryId id,
    in Identifier name,
    in StructMemberSeq members
  );

  TypeCode create_interface_tc (
    in RepositoryId id,
    in Identifier name
  );
```

```
    TypeCode create_string_tc (
      in unsigned long bound
    );

    TypeCode create_wstring_tc (
      in unsigned long bound
    );

    TypeCode create_fixed_tc (
      in unsigned short digits,
      in short scale
    );

    TypeCode create_sequence_tc (
      in unsigned long bound,
      in TypeCode element type
    );

    TypeCode create_recursive_sequence_tc (
      in unsigned long bound,
      in unsigned long offset
    );

    TypeCode create_array_tc (
      in unsigned long length,
      in TypeCode element_type
    );
  };
};
```

1.15.6 Portable Object Adapter

```
module PortableServer{
 // forward reference
 interface POA;

 native Servant;

 typedef sequence<octet> ObjectId;

 exception ForwardRequest
 {
  Object forward_reference;
 };

 // ***********************************************
 //
 // Policy interfaces
 //
 // ***********************************************
 enum ThreadPolicyValue {
  ORB_CTRL_MODEL,
  SINGLE_THREAD_MODEL
 };
 interface ThreadPolicy : CORBA::Policy
 {
  readonly attribute ThreadPolicyValue value;
 };

 enum LifespanPolicyValue {
  TRANSIENT,
  PERSISTENT
 };
 interface LifespanPolicy : CORBA::Policy
 {
  readonly attribute LifespanPolicyValue value;
 };

 enum IdUniquenessPolicyValue {
  UNIQUE_ID,
  MULTIPLE_ID
 };
 interface IdUniquenessPolicy : CORBA::Policy
 {
  readonly attribute IdUniquenessPolicyValue value;
 };

 enum IdAssignmentPolicyValue {
  USER_ID,
  SYSTEM_ID
```

```
};

interface IdAssignmentPolicy : CORBA::Policy
{
  readonly attribute IdAssignmentPolicyValue value;
};

enum ImplicitActivationPolicyValue {
  IMPLICIT_ACTIVATION,
  NO_IMPLICIT_ACTIVATION
};

interface ImplicitActivationPolicy : CORBA::Policy
{
  readonly attribute ImplicitActivationPolicyValue value;
};

enum ServantRetentionPolicyValue {
  RETAIN,
  NON_RETAIN
};

interface ServantRetentionPolicy : CORBA::Policy
{
  readonly attribute ServantRetentionPolicyValue value;
};

enum RequestProcessingPolicyValue {
  USE_ACTIVE_OBJECT_MAP_ONLY,
  USE_DEFAULT_SERVANT,
  USE_SERVANT_MANAGER
};

interface RequestProcessingPolicy : CORBA::Policy
{
  readonly attribute RequestProcessingPolicyValue value;
};

// ***************************************************
//
// POAManager interface
//
// ***************************************************

interface POAManager
{
  exception AdapterInactive{ };

  void activate( )
    raises( AdapterInactive );
  void hold_requests( in boolean wait_for_completion )
```

```
    raises( AdapterInactive );
  void discard_requests( in boolean wait_for_completion )
    raises( AdapterInactive );
  void deactivate( in boolean etherealize_objects,
                   in boolean wait_for_completion )
    raises( AdapterInactive );
};

// ************************************************
//
// AdapterActivator interface
//
// ************************************************

interface AdapterActivator
{
  boolean unknown_adapter( in POA parent, in string name );
};

// ************************************************
//
// ServantManager interface
//
// ************************************************

interface ServantManager
{ };

interface ServantActivator : ServantManager {
  Servant incarnate (
    in ObjectId oid,
    in POA   adapter )
      raises ( ForwardRequest );

  void etherealize (
    in ObjectId oid,
    in POA   adapter,
    in Servant  serv,
    in boolean  cleanup_in_progress,
    in boolean  remaining_activations );
};

interface ServantLocator : ServantManager {
  native  Cookie;

  Servant preinvoke(
    in ObjectId     oid,
    in POA          adapter,
    in CORBA::Identifier  operation,
    out Cookie      the_cookie )
      raises ( ForwardRequest );
```

```
  void postinvoke(
     in ObjectId     oid,
     in POA          adapter,
     in CORBA::Identifier  operation,
     in Cookie       the_cookie,
     in Servant      the_servant );
};

// ***************************************************
//
// POA interface
//
// ***************************************************

interface POA
{
  exception AdapterAlreadyExists {};
  exception AdapterInactive { };
  exception AdapterNonExistent { };
  exception InvalidPolicy { unsigned short index; };
  exception NoServant { };
  exception ObjectAlreadyActive { };
  exception ObjectNotActive { };
  exception ServantAlreadyActive { };
  exception ServantNotActive { };
  exception WrongAdapter { };
  exception WrongPolicy { };

  //--------------------------------------------------
  //
  // POA creation and destruction
  //
  //--------------------------------------------------

  POA create_POA( in string adapter_name,
  in POAManager a_POAManager,
  in CORBA::PolicyList policies )
    raises ( AdapterAlreadyExists, InvalidPolicy );

  POA find_POA( in string adapter_name, in boolean activate_it )
    raises ( AdapterNonExistent );

  void destroy( in boolean etherealize_objects,
        in boolean wait_for_completion );

  // ************************************************
  //
  // Factories for Policy objects
  //
  // ************************************************
```

```
~~ThreadPolicy~~
  ~~create_thread_policy( in ThreadPolicyValue value );~~

LifespanPolicy
  create_lifespan_policy( in LifespanPolicyValue value );

IdUniquenessPolicy
  create_id_uniqueness_policy
    ( in IdUniquenessPolicyValue value );

IdAssignmentPolicy
  create_id_assignment_policy
    ( in IdAssignmentPolicyValue value );

~~ImplicitActivationPolicy~~
  ~~create_implicit_activation_policy~~
    ~~( in ImplicitActivationPolicyValue value );~~

~~ServantRetentionPolicy~~
  ~~create_servant_retention_policy~~
    ~~( in ServantRetentionPolicyValue value );~~

~~RequestProcessingPolicy~~
  ~~create_request_processing_policy~~
    ~~( in RequestProcessingPolicyValue value );~~

//--------------------------------------------------
//
// POA attributes
//
//--------------------------------------------------

readonly attribute string   the_name;
readonly attribute POA      the_parent;
readonly attribute POAManager the_POAManager;
~~attribute AdapterActivator   the_activator;~~

//--------------------------------------------------
//
// Servant Manager registration:
//
//--------------------------------------------------

~~ServantManager get_servant_manager( )~~
  ~~raises ( WrongPolicy );~~

~~void set_servant_manager( in ServantManager imgr )~~
  ~~raises ( WrongPolicy );~~

//--------------------------------------------------
//
```

```
// operations for the USE_DEFAULT_SERVANT policy
//
//------------------------------------------------

~~Servant get_servant( )~~
 ~~raises ( NoServant, WrongPolicy );~~

~~void set_servant( in Servant p_servant )~~
 ~~raises ( WrongPolicy );~~

// ************************************************
//
// object activation and deactivation
//
// ************************************************

ObjectId activate_object( in Servant p_servant )
 raises ( ServantAlreadyActive, WrongPolicy );

 void activate_object_with_id(
   in ObjectId id,
   in Servant  p_servant )
 raises ( ServantAlreadyActive, ObjectAlreadyActive,
   WrongPolicy );

void deactivate_object( in ObjectId oid )
 raises ( ObjectNotActive, WrongPolicy );

// ************************************************
//
// reference creation operations
//
// ************************************************

Object create_reference (
   in CORBA::RepositoryId intf )
 raises ( WrongPolicy );

Object create_reference_with_id (
   in ObjectId       oid,
   in CORBA::RepositoryId  intf )
 raises ( WrongPolicy );

//------------------------------------------------
//
// Identity mapping operations:
//
```

```
    //--------------------------------------------------

    ObjectId servant_to_id( in Servant p_servant )
      raises ( ServantNotActive, WrongPolicy );

    Object servant_to_reference( in Servant p_servant )
      raises ( ServantNotActive, WrongPolicy );

    Servant reference_to_servant( in Object reference )
      raises ( ObjectNotActive, WrongAdapter, WrongPolicy );

    ObjectId reference_to_id( in Object reference )
      raises ( WrongAdapter, WrongPolicy );

    Servant id_to_servant( in ObjectId oid )
      raises ( ObjectNotActive, WrongPolicy );

    Object id_to_reference( in ObjectId oid )
      raises ( ObjectNotActive, WrongPolicy );

  };

  // ***************************************************
  //
  // Current interface
  //
  // ***************************************************

  interface Current : CORBA::Current
  {
    exception NoContext { };

    POA get_POA( ) raises ( NoContext );
    ObjectId get_object_id( ) raises ( NoContext );
  };

};
```

1.15.7 Interceptors

As Interceptors are omitted from minimumCORBA, all of the CORBA IDL for Interceptors, as defined in chapter 18 of the CORBA specification, is omitted from minimumCORBA IDL.

END OF DOCUMENT