Date: 31 May 2005

CORBA Reflection

ptc/2005-05-04

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

IONA Technologies, PLC

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER

DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

IONA Technologies, PLC

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents & Specifications, Report a Bug/Issue.

1 Scope

This specification defines reflective operations for CORBA objects. Such operations provide a way for applications to obtain metadata describing an object's interface directly from the object itself, rather than requiring the presence of a separate metadata service such as an Interface Repository. This specification provides a general approach to metadata discovery based on object narrowing, and it supports multiple metadata formats in an easily extensible manner.

2 Conformance

This specification defines three conformance points. ORB implementations shall support both of the following conformance points:

- The **Reflection** IDL module and its contents.
- The retrieval of **Reflection::XMLFormatter** objects from **ORB::resolve_initial_references**.

ORB implementations may also optionally support the automatic generation of reflection implementation code for static skeletons as part of the IDL compilation process.

Details for each of these conformance points can be found in Section 7, "CORBA Reflection."

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply.

- "Common Object Request Broker Architecture: Core Specification," OMG document formal/02-12-06.
- IFR Meta Object Facility (MOF) model, OMG document mars/2004-08-14.
- "Meta Object Facility (MOF) 2.0 XMI Mapping Specification," OMG document ptc/03-11-04.
- IFR XML schema, OMG document mars/2004-08-15. Note that for convenience of evaluating the IFR XML schema that is part of this RFC, document mars/2004-08-15 (a zip file) also contains the non-normative file named XMI.xsd, which is a subset of the XML schema from section 2.2.2 of the "XML Metadata Interchange (XMI) Specification," OMG document formal/03-05-02.
- "Deployment and Configuration of Component-based Distributed Applications Specification," OMG document ptc/04-03-10.

4 Terms and Definitions

For the purposes of this specification, the terms and definitions given in the normative reference and the following apply.

- *Reflection:* sometimes called *introspection,* an approach that allows applications to obtain information about objects and interfaces at runtime and use that information to perform dynamic invocations on those objects.

- *Target object*: in the context of a request made by a client application, the CORBA object that is the target of that request.

5 Symbols

This section is not applicable to this document.

6 Additional Information

6.1 Changes to Adopted OMG Specifications

This specification extends the CORBA 3.0.2 specification (formal/2002-12-06). No changes to existing CORBA features or constructs are made by this specification; rather, this specification contains pure extensions that build on top of existing CORBA features and constructs. The only addition that might be viewed as an exception to this is that this specification adds one new initial reference retrievable from the ORB's **resolve_initial_references** operation. This specification is intended to be added to a future version of the CORBA specification as a new chapter.

6.2 How to Read this Specification

The rest of this document contains the technical content of this specification. Paragraphs marked as editorial comments are not normative; they are provided only to help explain design choices, trade-offs, and future work items.

6.3 Implementation

The approach to adding reflection support to CORBA objects described in this document is implemented in IONA Technologies' Orbix® product, version 6. The primary difference between the Orbix implementation and the approach defined in this document is that Orbix uses a proprietary manually-coded XML schema to represent object metadata.

6.4 Acknowledgements

If you have comments or questions about this RFC, please contact Steve Vinoski of IONA Technologies at vinoski@iona.com.

Thanks to Frank Pilhofer of Mercury Computer Systems, Inc., for his assistance with defining the IFR metadata provider interface, and also for developing the IFR MOF model and generating the IFR XML schema from it. Thanks also to Rebecca Bergersen of IONA Technologies and Frank Pilhofer for reviewing drafts of this document.

7 CORBA Reflection

7.1 Overview

Reflection, sometimes called *introspection*, allows applications to obtain information about objects and interfaces at runtime and use that information to perform dynamic invocations on those objects. This information about objects and interfaces, commonly referred to as *metadata*, includes details about the number of attributes and operations in an interface, their signatures, and the definitions of all the types used in those signatures.

The Interface Repository (IFR) provides facilities for obtaining object metadata at runtime, but in practice, the IFR is useful only for certain applications. Specifically, such applications must explicitly be deployed together with an IFR, and that IFR must be populated with the metadata for all the application's objects that might be of interest to other applications. In practice, applications deployed into production are rarely accompanied by an IFR, for at least two reasons. One reason is that the typical IFR implementation requires one or more processes that run separately from the application, thus adding the need for extra computing resources, administration, and maintenance to the production system. Another reason is that most CORBA applications use static method invocation, and thus do not require access to an IFR.

While static CORBA applications can operate without an IFR, once they are deployed the lack of an IFR can make such applications difficult to integrate later with dynamic CORBA applications and with other non-CORBA middleware approaches such as Web Services. Without an IFR, the CORBA application metadata is neither available nor programmatically discoverable, thus hampering integration efforts that are typically intended to enhance or reuse existing applications.

Ironically, all CORBA objects, whether implemented with static skeletons or via the Dynamic Skeleton Interface (DSI), already contain reflection metadata. In fact, without it, it would be impossible for an ORB implementation to properly dispatch incoming requests to target objects. The reflection facilities described here provide access to this existing metadata via regular CORBA object narrowing and operation invocation techniques.

7.2 General Design

There are two elements of this reflection design: one that allows an object to provide its own reflection metadata, and one that allows a client to ask an object for its reflection metadata. CORBA objects that provide direct access to their metadata support one or more *reflection provider* interfaces. Reflection providers can be implemented either by code generated during the IDL compilation process (for static CORBA applications, based on static skeletons) or by implementing the servant to call ORB-supplied metadata formatting functions (for dynamic CORBA applications, based on the DSI). Clients can determine whether a given object supports CORBA reflection by attempting to narrow the object's reference to the desired reflection provider interface. If the narrow succeeds, the client can use the resulting object reference to obtain the object's metadata. If the narrow fails, then the object does not support such introspection.

One key design goal for CORBA reflection is to allow for straightforward metadata retrieval by non-CORBA clients. Because of the variety of middleware deployed in production environments, it is not uncommon that disparate middleware systems require integration. For example, a business might want to extend services implemented using CORBA to another part of its enterprise that uses Web Services. To satisfy this goal, the fundamental CORBA reflection interfaces are designed to be relatively simple, avoiding data types and approaches that do not map well to other middleware systems.

7.2.1 Reflection Metadata Provider Interface

A reflection provider interface[1] supports operations for metadata retrieval. Each different operation in the reflection provider interface returns metadata in a particular format.

This specification defines two reflection provider operations: one that returns its metadata in an XML format, and one that returns its metadata as an **any** containing ~~a **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription**~~ an instance of an IFR interface description structure. Each of these operations is described in more detail below.

All standard reflection provider definitions reside within the **Reflection** module. The full **Reflection** module is shown in Appendix A, and is also available in the separate Reflection.idl file (OMG document ~~mars/2004-07-05~~ptc/2005-05-05).

7.2.2 IFRProvider Interface

The **IFRProvider** interface defined in the **Reflection** module supplies two operations for retrieving metadata from an object.

```
module Reflection {
    typeprefix Reflection "omg.org"
    exception FormatNotSupported {};
    exception TypeNotSupported {};

    interface IFRProvider {
        any omg_get_ifr_metadata(
            in CORBA::RepositoryId metadata_type
        ) raises(FormatNotSupported, TypeNotSupported);
        string omg_get_xml_metadata(
            in CORBA::RepositoryId metadata_type
        ) raises(FormatNotSupported, TypeNotSupported);
    };
};
```

Because the **IFRProvider** interface is mixed into application-defined interface inheritance hierarchies, each operation name is prefixed with “omg_” to help keep it out of application name space.

Applications may wish to support only one of the **IFRProvider** metadata retrieval operations. For such cases, the unsupported metadata retrieval operation shall be implemented to raise the **FormatNotSupported** exception.

The omg_get_ifr_metadata Operation

The **omg_get_ifr_metadata** operation allows CORBA clients to retrieve metadata directly from an object in the form of ~~the IFR **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** type~~ one of the IFR interface description types. To allow reflection to work properly between clients and servers based on different versions of CORBA, the client specifies the typeid of the interface description type it is requesting as the **metadata_type** argument to the **omg_get_ifr_metadata** operation.

- Clients pass the typeid **“IDL:omg.org/CORBA/InterfaceDef/FullInterfaceDescription:1.0”** when requesting metadata in the form of the **CORBA::InterfaceDef::FullInterfaceDescription** type.

---

1. Note that the use of the term “provider” in this context is not the same as its use within the CORBA Component Model.

- Clients pass the typeid **"IDL:omg.org/CORBA/InterfaceAttrExtension/ ExtFullInterfaceDescription:1.0"** when requesting metadata in the form of the **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** type.

~~This type~~Each of these types can describe all metadata needed to perform dynamic invocations on an object, but does so using pure IDL data types and without requiring the presence or use of an IFR.

If a server does not support the type requested by the **metadata_type** argument, it raises the **TypeNotSupported** exception. This can occur, for example, if a CORBA 3.x client requests a **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** from a CORBA 2.x server.

The return type of the **omg_get_ifr_metadata** operation is **any** rather than **CORBA::InterfaceDef::FullInterfaceDescription** or **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** to allow static reflection skeletons to avoid dependencies on the IFR definitions. These definitions are quite extensive, and requiring them could make the compilation or deployment of such skeletons prohibitive for some deployment situations. The use of **any** also allows reflection implementations to support multiple versions of CORBA, by returning metadata in the form requested by the client. ~~An instance of the **any** type returned from **omg_get_ifr_metadata** shall always contain an instance of the **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** type.~~

Conforming reflection implementations shall set the **CORBA::IDLType** object reference fields nested within the **CORBA::InterfaceDef::FullInterfaceDescription** or **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** instance returned from the **omg_get_ifr_metadata** operation, such as the **type_def** fields present in IFR types such as **ValueMember** and **StructMember**, to nil. These fields normally refer to objects hosted by an IFR instance, and thus they are not useful or required within a reflection context.

The omg_get_xml_metadata Operation

The **omg_get_xml_metadata** operation returns an XML document as a **string**. The returned XML document conforms to an XML schema derived from the **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** IDL type, and thus contains exactly the same information as **ExtFullInterfaceDescription**. (This schema also supports the **CORBA::InterfaceDef::FullInterfaceDescription** type, since that type differs from a **ExtFullInterfaceDescription** only in a minor way where attributes are concerned, but the schema still requires minor editorial work for the final FTF report.) This XML schema, which can be found in OMG document mars/2004-08-15, is the result of reverse-engineering the subset of the IFR IDL definitions used in the **ExtFullInterfaceDescription** type into a Meta Object Facility (MOF) model, which can be found in OMG document mars/2004-08-14, and then applying to that model the XML schema production rules from the "Meta Object Facility (MOF) 2.0 XMI Mapping Specification" (OMG document ptc/03-11-04). Most IDL types map cleanly into XML schema, but because the **CORBA::TypeCode** and **any** types do not have a straightforward mapping into XML schema, they use the same representation as in "Deployment and Configuration of Component-based Distributed Applications Specification" (OMG document ptc/04-03-10).

XML Reflection Metadata Example

An object supporting the following simple IDL interface

```
interface HelloWorld {
    void hello(in string msg);
};
```

produces the following XML reflection metadata:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<InterfaceRepository:ExtFullInterfaceDescription
  xmlns:InterfaceRepository="http://schema.omg.org/spec/IFR/1.0/"
  xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schema.omg.org/spec/IFR/1.0/ IFR.xsd">
      <name>HelloWorld</name>
      <id>IDL:HelloWorld:1.0</id>
      <defined_in>::</defined_in>
      <version>1.0</version>
      <operation>
          <name>hello</name>
          <id>IDL:HelloWorld/hello:1.0</id>
          <defined_in>::HelloWorld</defined_in>
          <version>1.0</version>
          <mode>OP_NORMAL</mode>
          <result>
              <kind>tk_void</kind>
          </result>
          <parameter>
              <name>msg</name>
              <mode>PARAM_IN</mode>
              <type>
                  <kind>tk_string</kind>
              </type>
          </parameter>
      </operation>
      <type>
          <kind>tk_objref</kind>
          <objref>
              <name>HelloWorld</name>
              <typeId>IDL:HelloWorld:1.0</typeId>
          </objref>
      </type>
</InterfaceRepository:ExtFullInterfaceDescription>
```

A more complicated example is shown below:

```
interface B {
    struct S {
        long m1;
        sequence<S> m2;
    };
    exception NotFound {};
    exception NotSupported {};

    S get_value(in long key) raises(NotFound, NotSupported);
};
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<InterfaceRepository:ExtFullInterfaceDescription
  xmlns:InterfaceRepository="http://schema.omg.org/spec/IFR/1.0/"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schema.omg.org/spec/IFR/1.0/ IFR.xsd">
      <name>B</name>
      <id>IDL:B:1.0</id>
      <defined_in>::</defined_in>
      <version>1.0</version>
      <operation>
          <name>get_value</name>
```

```
<id>IDL:B/get_value:1.0</id>
<defined_in>::B</defined_in>
<version>1.0</version>
<mode>OP_NORMAL</mode>
<result>
    <kind>tk_struct</kind>
    <struct xmi:id="B.S">
        <name>S</name>
        <typeId>IDL:B/S:1.0</typeId>
        <member>
            <name>m1</name>
            <type>
                <kind>tk_long</kind>
            </type>
        </member>
        <member>
            <name>m2</name>
            <type>
                <kind>tk_sequence</kind>
                <sequence>
                    <elementType>
                        <kind>tk_struct</kind>
                        <struct href="#B.S">
                            <typeId>IDL:B/S:1.0</typeId>
                        </struct>
                    </elementType>
                </sequence>
            </type>
        </member>
    </struct>
</result>
<parameter>
    <name>key</name>
    <mode>PARAM_IN</mode>
    <type>
        <kind>tk_long</kind>
    </type>
</parameter>
<exception>
    <name>NotFound</name>
    <id>IDL:B/NotFound:1.0</id>
    <defined_in>::B</defined_in>
    <version>1.0</version>
    <type>
        <kind>tk_except</kind>
        <struct>
            <name>NotFound</name>
            <typeId>IDL:B/NotFound:1.0</typeId>
        </struct>
    </type>
</exception>
<exception>
    <name>NotSupported</name>
    <id>IDL:B/NotSupported:1.0</id>
    <defined_in>::B</defined_in>
    <version>1.0</version>
    <type>
        <kind>tk_except</kind>
        <struct>
            <name>NotSupported</name>
            <typeId>IDL:B/NotSupported:1.0</typeId>
```

```
                </struct>
            </type>
        </exception>
    </operation>
    <type>
        <kind>tk_objref</kind>
        <objref>
            <name>B</name>
            <typeId>IDL:B:1.0</typeId>
        </objref>
    </type>
</InterfaceRepository:ExtFullInterfaceDescription>
```

7.3 Using Reflective Objects

Client applications that wish to retrieve metadata from a CORBA object using one of the standard reflection provider interfaces simply perform the following steps:

1. Narrow the object reference for the target object to **IFRProvider**.
2. If the narrow fails, the target object does not support standard metadata retrieval.
3. If the narrow succeeds, invoke the desired metadata retrieval operation (**omg_get_xml_metadata** for metadata in XML form, or **omg_get_ifr_metadata** for metadata in the form of a **CORBA::InterfaceDef::FullInterfaceDescription** or **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** structure) to obtain the target object's metadata.

7.4 Implementing Reflective Objects

Because **IFRProvider** is just a normal IDL interface, it can be implemented using normal CORBA object implementation techniques. One simply derives the target object's most derived interface from the **IFRProvider** interface and implements its operations.

While implementing CORBA reflection in the manner described above is doable, it leaves much to be desired. First, it forces object implementers to provide their own formatting code that converts object metadata in the form of an **ExtFullInterfaceDescription** structure into XML. Second, it forces implementers to go back to their original IDL, modify the inheritance hierarchy to introduce the desired reflection provider interfaces, and modify their server implementation to add the reflection provider operation implementations.

It is possible to overcome most of these limitations for many applications. To address the formatting problem, the ORB shall provide a local object to assist with XML formatting. This formatting object is especially useful for DSI-based applications. The issue of having to modify the IDL inheritance hierarchy can be addressed for static applications through code generation, by building support for generating reflection implementation code into the IDL compiler. Details on these approaches are explained below.

7.4.1 Formatting Facilities

To eliminate the need for object implementers to write their own metadata formatting functions, the ORB shall provide a local metadata formatting object. The **XMLFormatter** interface is shown below.

```
module Reflection {
    typeprefix Reflection "omg.org"
```

```
        local interface XMLFormatter {
            string format_metadata(in any intf_desc);
        };
};
```

The **format_metadata** operation takes an **any** holding a **CORBA::InterfaceDef::FullInterfaceDescription** or **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** structure and returns a string containing XML-formatted metadata conforming to the XML schema defined in OMG document mars/2004-08-15. If the **any** instance passed into the **format_metadata** operation contains no value or contains a value of a type other than **CORBA::InterfaceDef::FullInterfaceDescription** or **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription**, **format_metadata** raises a **CORBA::BAD_PARAM** exception. An **any** is used as the parameter type to ensure that ORB implementations can be decoupled from the Interface Repository IDL definitions.

An application can obtain an **XMLFormatter** object from the ORB by passing the string "XMLReflectionFormatter" to its **resolve_initial_references** operation.

An important element of reflection implementation to note is that the object metadata returned from reflection provider operations need not contain metadata for the reflection provider interfaces themselves. This is because calling applications are aware of the reflection provider interfaces *a priori*, otherwise they would not be able to narrow to the reflection provider interfaces and invoke the metadata retrieval operations.

7.4.2 Static Reflective Operations

Static server applications generally rely on skeletons generated via an IDL compilation process. It is possible to augment such skeletons with full implementations of the **IFRProvider** interface, thus relieving server implementers from having to develop their own reflective operation implementations for these interfaces.

Automatic generation of **IFRProvider** requires the IDL compiler to implicitly insert them as base interfaces. For an interface that has no base classes, this is easy; the IDL compiler simply treats the reflection provider interface as a base interface. For interfaces that already have base interfaces, the IDL compiler inserts the reflection provider interface as a base interface at the base of the inheritance hierarchy. For example, consider the classic inheritance "diamond" shown in Figure 1.

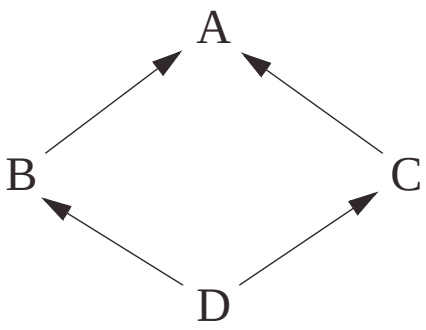

Figure 1. Diamond inheritance hierarchy.

Whether the object being implemented supports **A** or **D** as its most derived interface, the IDL compiler inserts the desired reflection provider interface into the inheritance hierarchy the same way: it introduces it as a base interface for interface **A**, as shown in Figure 2.

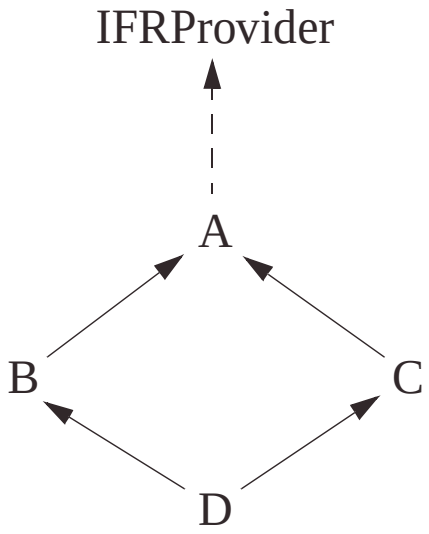

Figure 2. Diamond inheritance hierarchy with implied reflection provider inheritance.

Figure 2 shows the implied inheritance from the reflection provider interface introduced into the interface hierarchy by the IDL compiler. The compiler then proceeds to generate skeletons based on this modified inheritance hierarchy. It also generates implementations for the reflection provider interface methods. These can simply create appropriate **FullInterfaceDescription** or **ExtFullInterfaceDescription** structures and pass them to the standard formatter objects retrieved from **ORB::resolve_initial_references**, or they can call proprietary formatting functions. In Java, for example, the former approach is preferred in order to preserve skeleton portability, while the latter approach is typically required in C++ due to the fact that C++ servant base classes do not provide standard ORB accessor member functions.

7.4.3 Dynamic Reflective Applications

Since dynamic server applications by definition do not use static skeletons, they cannot rely on code generation techniques to provide reflection implementations for them. Thus, a DSI-based servant is responsible for implementing reflective operations. It does this by filling in the **CORBA::InterfaceDef::FullInterfaceDescription** or **CORBA::InterfaceAttrExtension::ExtFullInterfaceDescription** structure and returning it directly from its implementation of the **IFRProvider::omg_get_ifr_metadata** operation, or, to implement the **IFRProvider::omg_get_xml_metadata** operation, by passing the **FullInterfaceDescription** or **ExtFullInterfaceDescription** structure to an **XMLFormatter** object obtained from **ORB::resolve_initial_references** and returning the resulting XML string.

A Reflection IDL Module

The IDL definition below, which can also be found in OMG document number ~~mars/2004-07-05~~ptc/2005-05-05, was compiled successfully with the Orbix ~~6.0.1~~6.2 IDL compiler, except a #pragma prefix directive was used to set the "omg.org" type prefix instead of using the **typeprefix** keyword.

```
#include "orb.idl"

module Reflection {
    typeprefix Reflection "omg.org"
    exception FormatNotSupported {};
    exception TypeNotSupported {};

    interface IFRProvider {
        any omg_get_ifr_metadata(
            in CORBA::RepositoryId metadata_type
        ) raises(FormatNotSupported, TypeNotSupported);
        string omg_get_xml_metadata(
            in CORBA::RepositoryId metadata_type
        ) raises(FormatNotSupported, TypeNotSupported);
    };

    local interface XMLFormatter {
        string format_metadata(in any intf_desc);
    };
};
```