[Top] [Prev] [Next] [Bottom]

Dynamic Skeleton Interface

5

The *Dynamic Skeleton interface* (DSI) is a way to deliver requests from an ORB to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. This contrasts with the type-specific, OMG IDL-based skeletons, but serves the same architectural role.

DSI is the server side's analogue to the client side's *Dynamic Invocation Interface* (DII). Just as the implementation of an object cannot distinguish whether its client is using type-specific stubs or the DII, the client who invokes an object cannot determine whether the implementation is using a type-specific skeleton or the DSI to connect the implementation to the ORB.

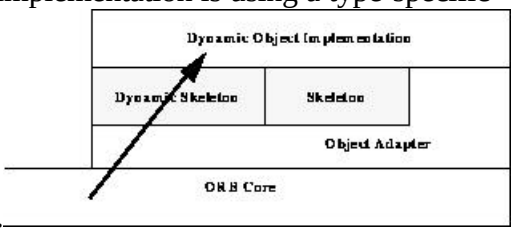

Figure 5-1 Requests are delivered through skeletons, including dynamic ones

DSI, like DII, has many applications beyond interoperability solutions. Uses include interactive software development tools based on interpreters, debuggers and monitors that want to dynamically interpose on objects, and support for dynamically-typed languages such as LISP.

5.1 Overview

The basic idea of the DSI is to implement all requests on a particular object by having the ORB invoke the same upcall routine, a *Dynamic Implementation Routine* (DIR). Since in any language binding all DIRs have the same signature, a single DIR could be used as the implementation for many objects, with different interfaces.

The DIR is passed all the explicit operation parameters, and an indication of the object that was invoked and the operation that was requested. The information is encoded in the request parameters. The DIR can use the invoked object, its object adapter, and the Interface Repository to learn more about the particular object and invocation. It can access and operate on individual parameters. It can make the same use of an object adapter as other object implementations.

The Dynamic Skeleton interface could be supported by any object adapter. Like type-specific skeletons, the DSI might have object adapter-specific details. This chapter describes a DSI interface for the Basic Object Adapter (BOA) and shows how it is mapped to C and C++.

5.2 Explicit Request State: ServerRequest Pseudo Object

The **ServerRequest** pseudo object captures the explicit state of a request for the DSI, analogous to the Request pseudo object in the DII. The following shows how it provides access to the information:

```
module CORBA {

pseudo interface ServerRequest

{

Identifier op_name ();

Context ctx ();

void params (inout NVList parms);

Any result ();

};

}
```

The target object of the invocation is provided by the language binding for the DIR. In the context of a bridge, it will typically be a proxy for an object in some other ORB.

The **op_name** operation returns the name of the operation being invoked; according to OMG IDL's rules, these names must be unique among all operations supported by this object's "most-derived" interface. Note that the opertion names for getting and setting attributes are **_get_<attribute_name>** and **_set_<attribute_name>**, respectively.

When the operation is not an attribute access, **ctx** will return the context information defined in OMG IDL for operation (if any). Otherwise, this context is empty.

Operation parameters will be retrieved with **params.** They appear in the NVList in the order in which they appear in the OMG IDL specification (left to right). This holds the "in", "out" and "inout" values.

The **result** operation is used to find where to store any return value for the call. Reporting of exceptions (which preclude use of result and out/inout values in **params**) is a function of the language mapping.

See each language binding for a description of the memory management aspects of these parameters.

5.3 Dynamic Skeleton Interface: Language

Mapping

Because DSI is defined in terms of a pseudo object, special attention must be paid to it in the language mapping. This section provides general information about mapping the Dynamic Skeleton Interface to programming languages.

Section 14.24, "Mapping of the Dynamic Skeleton Interface to C," on page 14-25 and Section 16.17, "Mapping of Dynamic Skeleton Interface to C++," on page 16-43 provide mappings of the Dynamic Skeleton Interface (supporting the BOA) to the C langugeand C++ languages.

5.3.1 ServerRequest's Handling of Operation Parameters

There is no requirement that a **ServerRequest** pseudo object be usable as a general argument in OMG IDL operations, or listed in "orb.idl".

The client side memory management rules normally applied to pseudo objects do not strictly apply to a ServerRequest's handling of operation parameters. Instead, the memory associated with parameters follows the memory management rules applied to data passed from skeletons into statically typed implementation routines, and vice versa.

In some language mappings, exceptions need special treatment. This is because the normal mapping for exceptions may require static knowledge of exception types. An example is the use of C++ exceptions, which require special run time typing information that can only be generated by a C++ compiler. Accordingly, the DSI and DII need an exception-reporting method that requires minimal compile-time support: the DIR needs to be able to provide the TypeCode for an exception as it reports the exception.

Finally, note that these APIs have been specified to support a performance model whereby the ORB doesn't implicitly consult an interface repository (i.e. perform any remote object invocations, potentially slowing down a bridge) in order to handle an invocation. All the typing information is *provided to* the **ServerRequest** pseudo object by an application. The ORB is allowed to verify that such information is correct, but such checking is not required.

5.3.2 Registering Dynamic Implementation Routines

Although it is not portably specified by previous CORBA specifications, any ORB and its BOA implementation must have some way of connecting type-specific skeletons to the methods that implement the operations. The Dynamic Skeleton interface uses the same mechanism.

A typical ORB/BOA implementation defines an operation, perhaps used when the object is activated, which specifies the methods to be used for a particular implementation class, for example, in C:

BOA_setimpl (BOA, ImplementationDef, MethodList, skeleton);

The MethodList would be the DIR; the skeleton could be a Dynamic Skeleton, which would construct a ServerRequest object and invoke the DIR with it.

Whatever mechanism, whether at link time, run time, and so forth, is used to bind ordinary

implementations to type-specific skeletons would also be used to bind dynamic implementations to dynamic skeletons. Such bindings could be maintained on a per-object, per-interface, per-class, or other basis.

[Top] [Prev] [Next] [Bottom]

---

pubs@omg.org