

LETTERS TO THE EDITOR

Dear CROSSTALK Editors:

Having read the November 2001 issue of CROSSTALK, I am somewhat at a loss concerning the section “Dynamic vs. Static Invocation” in the article *Factors to Consider When Selecting CORBA Implementations* by Dr. Thomas J. Croak. The author must have misunderstood the meaning of static invocation or he is using the term in a (for me) unknown way.

The author states that “static invocation can be used if the language, compiler, and operating system (and hardware) are known to be the same on both client and server.” This statement is of course true, but certainly static invocation can be used regardless of differences in client and server operating systems, compilers, and languages.

Also in the section “Questions for the CORBA ORB Salesman,” it is stated that “..., and you run the risk of future software failures given an operating system upgrade on portions of the architecture [when using static invocation].” I fail to see how this can be true, given that the CORBA architecture is specifically designed to be platform- and language-independent. Clients do not have to know, and indeed cannot know, implementation details of a server on the basis of the interface definition, and so will not be affected by server-side implementation changes.

According to my textbooks and to the best of my knowledge:

- Static invocation is used when you compile a client stub from Interface Definition Language and use that to contact the server skeleton. It is also known as early or compile-time binding. In other words, the interface (but not the implementation) is known beforehand.
- Dynamic invocation is used when the client does not know the interface in advance but queries the ORB for an interface (or method) definition. The client then builds a request based on the obtained definition and invokes it. This is known as late or run-time binding.

The difference between static invocation and dynamic invocation is very analogous to the difference between directly calling into vtable method pointers or using `IDispatch.Invoke` (after having called `IDispatch.GetIDsOfNames` and `IDispatch.GetTypeInfo` and so on ...) in COM.

Either way, the client will always be insulated from the server implementation details.

According to the glossary of the CORBA 2.5 specification (September 2001):

- Static invocation: Constructing a request at compile time. Calling an operation via a stub procedure.
- Dynamic invocation: Constructing and issuing a request whose signature is possibly not known until run-time.

Best Regards,

— Jan Holst Jensen

Dear CROSSTALK Editors:

The term “invocation” is overloaded to mean both the determination of the interface representation and the actual method call. The reader is using the first meaning of the term while I was using the second, and specifically stated so. The interface representation can be determined at compile time (referred to in the CORBA specification as Static Invocation Interface) or at run-time (referred to as Dynamic Invocation Interface in the CORBA specification). The method call on an object can be “invoked” dynamically or statically. In this sense of the term, the CORBA specification as well as other texts are not particularly clear. The interpretation I was using is that a dynamically invoked method call implies that the data will be marshaled, and a statically invoked method call implies that the data will not be marshaled. A danger of such ambiguity is that it leaves certain points open to the interpretation of the object request broker (ORB) provider, who may not implement the ORB consistently with the specification's intention.

The standpoint of my article dealt with safety-critical embedded and command-and-control systems typical of those found in Department of Defense combat systems where a function must be deterministic. Dynamic invocation, in either sense of the term would never be considered to be deterministic. Static invocation can be deterministic subject to the actual construction of the operating system, the compilers, and the ORB itself. However, the static invocation can become non-deterministic should any of these change. When I said, “Static invocation can be used if the

language, compiler and operating system (and hardware) are known to be the same on both client and server,” I should have said, “Static invocation can be used *safely* if ...”

Speaking from experience, I can attest that the CORBA architectural philosophy of platform and language independence has not been faithfully extended to all ORB implementations, for all platforms and all languages. If you have an environment of mixed operating systems and languages, it is often difficult to find a single ORB vendor for the whole environment. If you must use multiple vendors' products, you will rarely have interoperability problems with the standard data types such as integer, real, or string. Try passing a scalar array or a covariance matrix and there will often be interoperability problems. When I said, “You run the risk of future software failure given an operating system upgrade on portions of the architecture,” I was pointing out that an ORB supplier might make design decisions based on specific operating system functionality. If the underlying functionality of the operating system changes, the ORB's behavior may also change. When an ORB is ported to a different operating system or operating system version, testing does not always uncover all the behavior nuances, which may ultimately affect performance.

I feel strongly that any architect or designer should carefully consider the implications of any design decision on the expected behavior of the system as a whole, as well as the potential for unexpected changes in behavior given typical system evolution in compilers, operating systems, and commercial off-the-shelf products.

— Tom J. Croak