
Vancouver, British Columbia, Canada
5 – 10 October 1992

Addendum
to the
Proceedings

# Poster Submission—
# Developing Language Neutral Class Libraries With The System Object Model (SOM)

**Report by:**

Mike Conner
Nurcan Coskun
Scott Danforth
Larry Loucks
Andy Martin
Larry Raper
Roger Sessions
IBM Corporation

Object-oriented programming is quickly establishing itself as an important methodology in developing high quality, reusable code. The most promising packaging technology for reusable object-oriented code seems to be based on class libraries. Current technologies for packaging class libraries have several problems, the most important of which is that they are highly language biased. Class libraries developed in one language cannot be used with other languages. For example, a class library developed in C++ cannot be used by a Smalltalk programmer, and a Smalltalk library is of no use to a Cobol programmer. The System Object Model (SOM) is a new packaging technology designed to address this and other packaging issues.

SOM defines a protocol boundary in which classes and objects can be registered irrespective of the object model in which they are developed. The design of this protocol boundary is such that objects and definitions registered in the boundary are naturally and efficiently accessible (and even subclassable) from a number of different programming languages. The architecture of this boundary and the supporting runtime and tools constitute SOM.

In addition to defining a language neutral object model, SOM offers a number of features considered important in the packaging of industrial strength class libraries. These include support for dynamic loading and linking of classes; the ability to update class libraries without requiring that client applications be recompiled; the ability to ship class libraries without source code; and the ability to use object-oriented technology from traditional procedural programming languages.

It is important not to erode the quality of object support in existing object-oriented languages with respect to syntax, performance, flexibility, or development support. That is, SOM is designed to work *with* existing object-oriented languages, not to replace them. One of the ways programmers use classes is through subclassing. Support for cross language subclassing is more difficult than cross language use, but this form of customization of reusable components is the major advantage of object technology over well structured procedure libraries, and is an important focus of SOM.

Most commonly used programming languages are not object-oriented and have difficulty using and creating class libraries. However the SOM tools, combined with the SOM runtime, provide support for these languages so that the use and definition of class libraries is quite natural. These SOM provided object-oriented extensions are not intended to eliminate the need for object-oriented languages. Object-oriented languages will continue to offer many benefits to object developers and users. The SOM extensions are meant to provide those who choose procedural languages to use object-oriented tools.

In the current version of SOM as released on OS/2 2.0, we provide full tool support for only C language bindings. For a description of using these C language bindings, see [Sessions and Coskun], [Coskun and Sessions] and [SOM]. We also have experimental C++ bindings, designs for Smalltalk bindings, and bindings to an experimental object-oriented version of REXX. We are working with a wide variety of language suppliers, both internally and externally to provide language bindings to as many languages as possible.

SOM is designed to augment existing object-oriented languages, not to compete with them. Object-oriented languages already have object models. For these languages, SOM offers a language neutral packaging technology and upwardly compatible binary libraries.

C++, for example, is a very efficient object-oriented language, with direct support for encapsulation, polymorphism, and inheritance. Many programmers are already familiar with the language, and a large body of C++ code is already in existence. C++ has an object model that is optimized to meet the needs of C++.

However C++ as a technology for packaging class libraries has some significant drawbacks that SOM can help address. The most important is that class libraries written in C++ can be used *only* from C++. C++ libraries cannot be used by such commercially important languages as Cobol and Fortran, or even by other object-oriented programming languages such as Smalltalk. SOM solves this problem by allowing C++ libraries to be packaged in a language neutral form that they can be used (and subclassed) from *any* language with SOM bindings.

Another problem with C++ class libraries is that their binary versions are not upwardly compatible. When a new C++ class library is released client code typically has to be fully recompiled, even if the changes are unrelated to public interfaces. This problem is discussed in detail in [Sessions]. When a C++ class library is packaged with SOM many changes can be made to the library source without requiring recompilation of client code. SOM allows methods to be moved up the class hierarchy, instance variables to be deleted, new methods or instance variables to be added, and new classes to be inserted into the class hierarchy. Any of these changes would require C++ clients to recompile their code, had they been using standard C++ class libraries.

The design of the SOM protocol boundary was faced with seemingly conflicting requirements. On the one hand it needs a great deal of flexibility to span the different method resolution, memory management and runtime support strategies employed by various different programming languages and object models. On the other hand, it must have little or no performance overhead in cases where, for example, a C++ program is using a class written in C++.

SOM achieves both of these goals by using language neutral objects in the protocol boundary. These objects present an interface that is either very abstract (saying nothing about method resolution for example) or very optimized depending on the view of the using program. For example, a C program might look up a method via an offset into a method procedure table while a Smalltalk program might access that same method via a dispatch function that completely hides the resolution mechanism.

A key aspect of this approach is the SOM Object Interface Definition Language (OIDL). OIDL provides a language neutral class definition and is the basis for the tools that support access to the protocol boundary. We have several compilers, one for each language binding. In the future, we are planning on supporting CORBA IDL [CORBA].

SOM provides three types of method dispatch mechanisms: offset lookup, name lookup and dispatch function.

The *offset lookup* is the fastest method dispatch mechanism and is designed to be used by the compiled, statically typed languages. This type of method dispatch exposes method signatures and some aspects of the class hierarchy.

The *name lookup* is the second fastest method dispatch mechanism and only exposes method signatures. The name lookup mechanism should be used when there is a requirement to hide all aspects of the class hierarchy. It is also useful when the only information known at compile time is the method signature.

The *dispatch function* mechanism has more overhead in comparison to offset lookup and name lookup techniques, but is much more flexible and can be used to send messages to the SOM objects dynamically. Dispatch resolution allows object providers

complete control over the resolution process, allowing even such resolution models as the one used in the Common Lisp Object System (CLOS), which dispatches based on the types of more than one method parameter. Dispatch resolution also allows object providers an efficient, and centralized point to perform parameter translation, memory management and other things that are necessary when interfacing a dynamically typed execution environment (such as Smalltalk or CLOS) to another execution environment.

The existing programming language bindings can support SOM through one of three techniques: macros, procedure wrappers and stub methods. Languages like C and C++ have powerful macro processors which can be used to extend them to support SOM. Languages like Pascal and Fortran do not have macro processors and instead use procedure wrappers for each method. The main function of the procedure wrapper is to hide method resolution computation. Dynamic Languages like Smalltalk and CLOS use stub methods which convert language specific object parameters into simple SOM values (int, double, etc.) as appropriate, and then wrap the return value in a language specific object. Stub methods can be very small and dynamically generated when a language makes use of the dispatch function mechanism for method resolution.

SOM has moved well beyond the research state. It is actively being used throughout IBM and has been accepted as the standard packaging technology for class libraries within IBM. It is part of the OS/2 2.0 toolkit and its release on AIX is anticipated. IBM has already shipped its first product based on SOM: the newly released Workplace Shell of OS/2 2.0. IBM is actively working with a wide variety of language vendors and industry standard groups to make SOM a non-proprietary and generally accepted technology.

## References

[CORBA] The Common Object Request Broker Architecture and Specification. OMG Document Number 91.12.1, Object Management Group, Framingham, Massachusetts.

[Coskun and Sessions] Class Objects in SOM, by Nurcan Coskun and Roger Sessions. OS2/2.

[Sessions] Class Construction in C and C++: *Object-Oriented Programming Fundamentals*, by Roger Sessions. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[Sessions and Coskun] Object-Oriented Programming in OS/2 2.0, by Roger Sessions and Nurcan Coskun. IBM Personal Systems Developer, Winter, 1992, 107–120.

[SOM] *OS/2 2.0 Technical Library System Object Model Guide and Reference*, IBM Doc S10G6309.

## Contact information:

Roger Sessions
I.B.M. Corporation
Object Technology Group
11400 Burnet Road
Austin, Texas 78758