The System Object Model (SOM) and the Component Object Model (COM):

A comparison of technologies summarized

IBM Corporation
Object Technology Products Group
Austin, Texas

SOM

Object enabling technology consisting of an advanced object model and a syntax-free OO runtime engine sufficient for implementing object-oriented programming languages. SOM solves the problems of cross-language access and fragile base classes allowing class libraries to be distributed and subclassed in binary form.

COM

Ostensibly, a “language-neutral” binary interface specification but primarily a set of rules that must be implemented in every application implementing interoperable “Windows Objects”.

Programming methodology

SOM:
Object-oriented

COM:
Object-based

Comments:
Based on widely accepted parlance (e.g. Wegner, OOPSLA 87), object oriented programming presumes implementation inheritance, A correct characterization of the methodology is important because it typifies the programming paradigm that will be found in applications built using the methodology. COM requires programmers to adopt an unfamiliar, proprietary programming model by redefining prevalent terms and techniques in object oriented programming. SOM, on the other hand, uses a programming model familiar to programmers who use object-oriented programming languages.

Relationship to languages

SOM:
Language-neutral. Objects may be written in C or C++. Smalltalk and Cobol bindings areunder development by vendors. Direct-to-SOM C++ products (available from Metaware and IBM) map the ANSI C++ language directly into SOM.

COM:
Language-neutral, function pointer table based. The user is responsible for constructing the function table. Objects may be written in C or C++. Writing objects in C is more involved because the function table must be constructed manually instead of the C++ trick of using a virtual base class. Interestingly, the C++ method of using a virtual base class for building the function table may not work properly with all C++ compilers - not all compilers lay out the vtable structure the same way that the Microsoft compiler does.

Comments:
COM's object model does not map into any OOPL and cannot be used as an OOP runtime engine by compilers. SOM can be used as a common binary runtime engine for C++ compilers and other OOPLs

Static Typing

SOM:
Objects statically typed via the class hierarchy. For static method invocations, the compiler checks that a class is of the appropriate type.

COM:
Individual interfaces are statically typed, not objects. A programmer deals with an interface pointer and must verify support for an operation at runtime.

Comments:
Static typing permits the language compiler to prevent type mismatch errors from showing up at runtime. Lack of static typing introduces more runtime failure modes. COM objects require runtime operations to ascertain which interfaces are supported by an object.

Distributed object support

SOM:
Provided by the DSOM framework.

COM:
Will be provided in the Cairo release in 1995.

Compound Document Support

SOM:
Provided by OpenDoc.

COM:
Provided by OLE

Method Resolution Mechanisms

COM:
User responsibility, requires 2 steps:

1. Use QueryInterface or instantiation to get Interface
2. Indirect function call via interface vector table

Polymorphism is not directly supported and must be emulated by user-written code using containment.

COM supplies no user intercept capability. The granularity of polymorphism is at the interface level only.

SOM:
3 Forms (offset, name, dispatch).
User intercept capability for all three (via somDispatch).
Granularity of polymorphism: interface level, method level, or user-defined (arbitrary).

Comments:
Except for object instantiation, no COM code is involved in interface selection or function invocation. In this area COM is only a set of rules, all code involved in interface selection or function invocation is user-written.

SOM, on the other hand, performs method resolution (i.e., interface selection and function invocation) completely automatically, without user-written code. SOM users may, however, optionally elect to participate in method resolution by overriding the somDispatch mechanism.

Inheritance

SOM:

Supports single, multiple and abstract (interface) inheritance. Complete encapsulation of base class from derived class. Subclass uses only the published interfaces of the base class. Versioning is handled automatically during class construction. SOM has several mechanisms such as somIsA, etc.that can be used to simulate aggregation

COM:

Traditional OO inheritance is not supported. Instead, containment and aggregation are offered as substitutes for reuse. Implementation inheritance is considered bad due to possible implementation dependencies between a base and derived class. IUnknown interface is frozen since it must be embedded within all other interfaces.

Comments:

Aggregation is essentially a manual technique entirely implemented by user-written code (COM is not involved in aggregation; it simply provides rules about what users must write). On the other hand, SOM's implementation inheritance is automatic and not subject to the "base/derived" class problem since (in general) the only the base class' public interface is available to the derived class. In addition, lack of inheritance at the COM level means implementors must cope with 2 different object models, one for components, and a different one at the implementation language level (since no language offers aggregation).

Functional composition mechanisms

SOM:

- Containment (or Delegation (user-written)).
- Inheritance.
- Metaclass selection.
- Derived metaclasses based on multiple inheritance.
- Behavior changes are made by overriding methods.

COM:

- Containment (or Delegation (user-written)).
- Aggregation, but all objects not required to support it.
- Behavior are made changes by front-ending with new code.

Comments:

COM capabilities require user discipline and adherence to protocol conventions. SOM mechanisms are richer and more automatic. Some functions cannot be augmented via COM's front-ending technique since methods internally invoked within an implementation do not interact with the front-end. Despite the overblown terminology, aggregation remains essentially just an optimized form of containment, allowing a user to forego the writing of the delegation code.

Versioning

SOM:

Automatic base/derived class version checking using major & minor numbers. Optional user-written versioning using major & minor numbers. Evolution of classes permits refactoring methods, additions to inheritance tree, changes in object size, additions to interfaces.

COM:

User-written versioning using major & minor numbers. Method implementation refactoring, changes in inheritance tree, or object size not visible thorough COM. Additions may NOT be made to interfaces. Old interfaces must be kept and a new interface that include all the old methods along with the newly added methods must be created.

Comments:

There is a dramatic difference in philosophy between SOM and COM here. SOM permits interfaces to evolve over time without breaking existing client programs, but COM requires proliferation of new interfaces while retaining all old interfaces. Also, in SOM versioning is essentially automatic, while a user responsibility in COM.

Complexity

SOM:

- One object reference needed per object.
- Reference counting may be used when desired and can be automated using metaclass programming
- Inheritance is statically declared.
- Local classes and interfaces do not require globally-unique persistent Ids. DSOM makes use of UUIDs.
- No special coding required to permit subsequent implementation inheritance.

COM:

- An interface reference is required for each object interface.
- All objects must be reference counted.
- Aggregation requires user-written code to implement.
- All classes and interfaces must have a globally-unique persistent Id.
- Interface implementations must be specially coded to permit subsequent aggregation.
- All object lifecycle is implemented by the programmer.

Object Lifecycle support

SOM:
All object lifecycle support provided including object creation, object destruction, object factories.

COM:
Object lifecycle is implemented by the programmer according to complex rules and reference counting protocol.

Robust object model

SOM is a "language-neutral" packaging technology for class libraries. It permits libraries of object classes written in various object-oriented or procedural languages to be distributed in binary form, either as part of an operating system, a vendor tools product, or an application. Client programs written to use SOM-based class libraries are insulated from dependencies on unintentional details of the object's implementation, so that the supplier of the class library may produce subsequent releases with fixes or enhanced capabilities without fear of breaking the client programs.

The COM model is predominately conceptual in nature. Little programmer support is provided. Type safety is the responsibility of the programmer.

Standards and platform coverage

SOM:
- Invented by IBM, but built on the OMG CORBA standard.
- Supports inheritance as built into all OOP development languages.
- Uses IDL -- OMG's Interface Description Language to define all objects.
- Direct-to-SOM C++ Compiler products map ANSI C++ language directly and transparently into SOM.
- Available now on OS/2, AIX, Windows, Mac System 7. Will likely be available in the future on other Unix's, Novell Netware, IBM's Workplace, MVS and OS/400 operating systems.

COM:
- Invented by Microsoft, not part of any standard.
- Introduces paradigms, such as containment and aggregation, which are not part of any programming language.
- Available now on Windows 3.1. Beta avail for Windows NT & Chicago. Promised but not yet available for Mac System 7.

Comments:
Incorporation of standards assures that widespread user requirements are being addressed and that compatible APIs and products may appear across a broad range of industry platforms. SOM is explicitly committed to support of industry standards while Microsoft argues that tracking standards is counterproductive. Microsoft has indicated that the CORBA specification is incomplete and evolving and that it is unlikely to support CORBA.

Unique Features

SOM:

It is easy to overlook the runtime component of the SOM as merely providing support for language neutrality and upward binary compatibility. But, in fact, SOM embodies an advanced object model with some very important and desirable capabilities including:

- Dynamic class construction
- Metaclasses
- Name Lookup & dispatch method resolution
- Single, multiple, & abstract inheritance
- Emitter framework
- User dispatch intercept

COM:

COM has comparatively impoverished functionality.