Object Orientation: A Natural Evolution for HL7

Wes Rishel
Wes Rishel Consulting
(510) 522-8135
wes@wes.win.net

Abstract

Object-oriented technology is having a significant impact on the analysis and programming techniques used in information systems. The same concepts are also providing the basis for standardizing interfaces among applications and components thereof. These specifications exist at both the conceptual level and in terms of the detailed level of specificity needed for "plug and play" cooperation. These specifications not only apply to functions developed using object-oriented tools, but also provide a convenient model for interfacing legacy applications. There is significant interest in providing a set of standards for exchanging healthcare data and invoking healthcare functionality using object-oriented approaches. Some probable requirements for these interfaces are described in this paper including exchanging data by query and update notification and interfacing with compound documents and other componentized functions. There are a number of competing specifications for object-oriented interfacing in the information system industry. A specific mapping is proposed from HL7 Version 2.2 to an abstract object model based on healthcare application objects and healthcare data objects. This abstract model would be independent of any specific specifications, and a significant part of any complete specification for any of them. It is further proposed that HL7 support special interest groups to develop implementation guides describing how to apply the abstract object model to any of the specific, detailed approaches.

Why O-O?

"Object Oriented" has become one of the significant catch-phrases in the industry, subject to all the misappropriation and other abuses of any significant trend and many passing fads. Despite the confusion on what it means, we assert that object-oriented programming (OOP) is in fact a significant trend. This is based on our own observation that C++, Objective C, and Smalltalk are the implementation language of choice for most new healthcare products that we have seen, particularly those that are being developed for a Graphical User Interface environment. Of these three, the author has found C++ to be the predominant tool.

OOP techniques can also be used to create interfaces to new functionality from programs that are not themselves written in object-oriented languages. Examples of this include the creation of custom controls for Visual BASIC programs in C++ and the creation of interfaces to legacy systems that are described in object-oriented terms.

Features, Benefits, and Definitions

The features and benefits of OOP haven been amply described. The fundamental feature is "code hiding," where-in the program statements that support a data structure are unknown to programmers who use that structure in their work. The data structure, along with its

programming methods, is known as an object. Programmers that use the object know only a formally defined set of methods for invoking procedures that perform operations on the data. This makes object-oriented programs more robust and maintainable, because team members working on part of an object-oriented programs cannot be impacted by most changes to an object they use.

This conceptual model of an object is illustrated by Figure 1.

The term *object* has itself been used differently in different OOP implementations. We use it here to mean the combination of data and program code that is bundled together and represented as an atomic concept to other parts of a program. This definition applies whether we are talking about the object as an abstraction or a specific incarnation in a running program. We also use the term *method* to describe the public program call which is the means by which the data and functions of the object are made available to code in other objects.

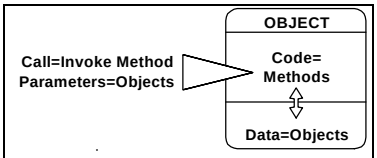

Figure 1. An object contains and hides its data, which is accessible through published methods or calls.

Another fundamental feature of OOP is "inheritance," whereby one object can be derived from another, inheriting most of its data definitions and methods. The derived object will generally differ from its predecessor in that it may have additional data and new methods. The derived object will frequently re-implement selected methods of its predecessor as well. One primary benefit of inheritance is that often drastically reduces the design and programming necessary to produce a new object. Indeed, one of the primary goals of Object Oriented Design (OOD) is determining the hierarchy of inherited objects that is most productive in terms of re-use and stable in the face of design changes.

Uniform Interfaces Lead to Significant Simplifications

These benefits can lead to dramatic simplification of implemented systems when the users of OOP environments follow a practice of establishing uniform interfaces to their objects. That is to say, where the designers give a common set of methods to objects that represent different things, the programs that manipulate those things can be drastically simplified. Any object that represents a printable document, for example, should have a "print myself" method with the identical name and parameters as any other. A "display myself" method can be implemented by diverse objects representing textual material, drawings, animations, and sound. Similarly, any object representing fundamental data (strings, numbers, arrays, etc.) should have a "format myself for display" method.

If a discipline of uniform interfaces is followed, the programs that manipulate those objects can use the same code to do the same function on these heterogeneous objects. It becomes easier, for example, to write a browser that has a function of displaying selected objects.

Other Uses of the Object-Oriented Metaphor

Object-orientation has had an impact on analysis techniques for applications as well. Although object-oriented analysis shares many common characteristics with entity-relationship analysis, it differs in that it offers the possibilities of inheritance and an integrated view of data and function. HL7's Quality Assurance Group and the Joint Working Group for a Common Data Model (in healthcare) have adopted Coad and Yourdon's approach to OOA for their underlying methodology.

The benefits of the object-oriented metaphor are being adopted at varying paces by the implementors of a number of operating systems including various Unix implementations, Microsoft Windows and Windows NT, NextSTEP, OS/2, and Taligent. Use of OOP is almost mandatory to develop the infrastructure necessary to provide a Graphical User Interface. "Drag and drop" operations among the visual objects on a desktop are implemented, for example, by having those objects make uniform object interfaces available to the window manager implementing the operation.

Compound Documents

One important use of the object-oriented metaphor is to implement compound documents. These are single documents composed of pieces created and maintained by different application programs. It is only practical to implement compound documents when a large set of uniform interfaces has been established for objects representing the components of the documents as they appear and are manipulated on secondary storage, the screen, and print media.

Indeed, the object models discussed below are often explained as if developing compound documents were their sole purpose. In fact, objects are used to implement much more than the compound document. By establishing uniform interfaces independently written applications can exchange data on demand or by notification of changes. They can also offer specific functions, such as creating drawings, performing complex calculations, retrieving specific information, etc. It would be easy to imagine applications providing specific functions like MedLine lookups, body surface area calculations, eligibility checking, rules-based searches of clinical databases and many other functions. Creating and marketing such objects will be greatly enhanced if there are standards to broaden the pool of potential users of the objects

Objects as Function Wrappers

The emphasis on data hiding and formal interfaces permits object interfaces to be used to interface applications that are not themselves written in an object-oriented language. As long as they can accept calls initiated by other objects, decode the objects that represent the parameters and encode the response data, the actual implementation language is of no concern.

Object Brokering

In environments with multiple, independently developed applications, *object brokering* systems are being used to implement inter-application messaging. This is true whether the applications reside on the same computer system or communicate across networks. Intersystem transactions passing data and invoking procedures are defined using the metaphors of object-oriented programming. Object brokering systems also facilitate establishing the association between applications by symbolic names, rather than more fragile mechanisms that rely on knowing where the objects that represent the application can be found in secondary storage or on a network.

The Common Object Group has published the Common Object Request Broker Architecture (CORBA), a detailed conceptual framework for implementing object brokering. It is notable in that it includes mechanisms for implementing object oriented interfaces among legacy applications written in diverse applications languages that are not object-oriented. CORBA is not a complete standard, in that it does not extend to the complete communications profiles and binary formats necessary to create interoperable implementations. It has, however, influenced a number of implementations of object brokering system.

A given implementation of an object broker can be described in terms of its CORBA compliance--the extent to which it includes the concepts of CORBA.

Complete Object Interface Specification = Objects + Interfaces + Protocols

In order to achieve the benefits of an object-oriented approach in practice, the cooperating applications must conform to a common specification that includes certain basic objects, their interfaces (common methods) and protocols. In this sense, a *protocol* is a specification of the circumstances in which methods can be invoked. If the Objects are the "Whats" and the Methods are the "Hows," the protocol is the collection of "Whens."

Each object brokering system, then, must have a complete object interface specification. Some complete object interface specifications that have been or are being implemented are shown in the table below.

Model[1]	Published By	Venue	Status
OLE2.0, based on the Component Object Model (COM)	Microsoft	Windows, Windows NT Macintosh (beta) Various Unix environments MVS, VMS, OSF and various other Unix Environments (future)	In production in OLE2.0. Some major applications available. Being implemented by Microsoft for Macintosh OS. Licensing agreements signed, no product announcements yet. A subset, called the Common Object Model to be interfaced by DEC to its Object Broker for connectivity to OSF and other OS environments.
System Object Model (SOM)	IBM	Windows, Windows NT, OS/2, Macintosh, various Unix, X-windows	In use in AIX and OS/2. Part of OpenDoc, expected to be released to developers in the Summer of 1994. Part of Taligent.
Distributed Objects	NeXT	NextStep HP/UX SunSOFT	In production for several years. Available for HP/UX in Portable Distributed Objects. To be available as part of Sun's Distributed Objects Everywhere (DOE)
Object Broker	DEC	VMS, MVS Various Unix	In production.
Common Object Services Specification (COSS)	SunSOFT	Solaris	Announcements pending.

Complete Object Interface Specifications Compared

These complete object interface specifications vary considerably on a number of parameters. OLE 2.0 can be considered the most widely available since Windows is at the most workstation seats, and a number of applications vendors have shipped products or made commitments to support it. Its Component Object Model does not use true inheritance, substituting a concept for re-use called aggregation [2]. Microsoft points out that the implementors of OLE 2.0 still have full use of inheritance within their implementation languages and that an advantage of its approach is that binding between cooperating applications occurs entirely at runtime. It claims that run-time inheritance across objects in independently implemented applications is inappropriate and will be fragile. Counter-claims in favor of SOM claim that fragility is not an issue.

OLE 2.0 does not currently support distribution across networks, although this has been announced and demonstrated. It is frequently characterized as dependent on C++, although implementation in other languages is feasible and will be easier as header libraries for those languages are developed and distributed. It is not CORBA compliant.

SOM supports inheritance and is "language neutral." It is CORBA-compliant and it is a part of the foundation of a number of current and future object environments including OpenDoc and Taligent. It appears to us, however, that SOM is a subset of the complete object interface specifications in OpenDoc and Taligent, so that SOM-compliance is not a guarantee of compatibility with either environment or of eventual interoperability between the environments.

The mechanisms to implement the language-neutral and inheritance features of SOM imply extra overhead at the time objects are instantiated. It will be some time before the impact of that overhead can be evaluated. There is considerable skepticism in the industry whether or how fast the various implementors of SOM-based object brokers and foundations can achieve any degree of interoperability [3]

Who Will Win The Great Complete Object Interface Specification Battle?

Adopting any one of these complete object interface specifications represents a significant investment for a software developer. Developers of healthcare applications (vendors and in-house) must make a fundamental strategic decision on complete object interface specification early in a project. The consequences of having to change are substantial. The factors effecting such a decision will include:

- √ the probable availability of other applications with which to interoperate at the end of the development cycle
- √ affiliation with the operating system of choice
- √ availability of software tools to support the chosen model
- √ availability of software engineers experienced in the chosen mode.
- √ the technical strengths and weaknesses of the complete object interface specification.

For HL7 the corresponding question might be which complete object interface specification must we support? In fact in this paper we recommend an approach where HL7 remains neutral with respect to this decision, while supporting its adoption in any of them.

Specific Issues Facing HL7

Historically, HL7 has adopted a policy of remaining independent of specific technologies such as communications protocols, operating systems, database technologies, and implementation languages. We have stood firm to support legacy technical environments in the face of criticism that we lacked the *savoir faire* of approaches that build on specific newer technologies. Why should we adapt ourselves any more to this particular new technology? In the author's opinion, there are several reasons.

- We have an "upstairs-downstairs" problem (figure 2). While we meet our traditional mission of sharing event-driven data among applications in a network, we have not provided support to applications sharing a workstation. Advances in Graphical User Interfaces and GUI technology have created this compelling need for sharing among workstation applications and those that are still running on dedicated systems.

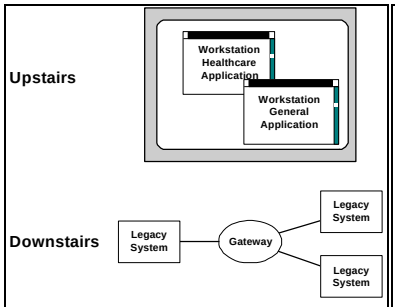

Figure 2. Solutions being implemented without benefit of standardization.

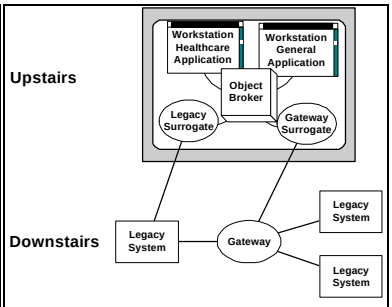

Figure 3. Solutions being implemented without benefit of standardization.

- The kinds of data exchanges necessary to support the workstation environment are being implemented now (figure 3) by many of the implementors that comprise our constituency.[4] While they are achieving a degree of interoperability among their own products they are losing any opportunity to use the same investments to achieve interoperability with other applications, because there is no standard way to implement healthcare data and events.

- There is need to support componentized application develop, where system builders will develop specific applications by piecing together components through object-oriented interfaces. Where there are multiple componentized healthcare applications

being assembled they must agree on the definition of the data objects that they take as data and return as results. (See figure 4.)

- There is a need to consider how healthcare objects interact with more general objects (e.g., compound documents) and determine if new standard objects or methods are needed.

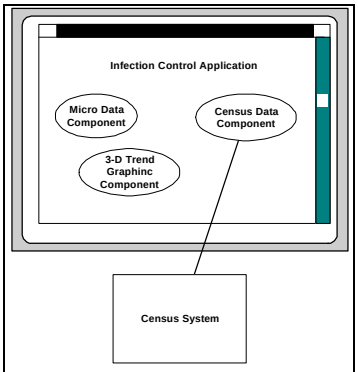

Figure 4. Objects used to implement componentized software.

Requirements Definition

What exactly are the requirements for object-oriented HL7? This is an interesting question. While we suggest some possibilities here, it clearly needs to be addressed by some canvas of our constituency.

(a) sharing of data about specific patients, encounters, orders, results, etc. among healthcare applications

(b) sharing of data about groups of patients, encounters, orders, results, etc. among healthcare applications

(c) notification of events (admissions, transfers, order entry, results available, etc.) among healthcare applications

(d) master file updating

(e) sharing of events that are peculiar to workstation applications (for example, enabling one application to share the identification of the patient the user has selected with another application)

(f) enabling healthcare applications to provide the ability to query and update data about patients encounters, orders, results, etc., through standard interfaces that are used by general applications in the system's complete object interface specification (for example, enabling an Excel user to write simple Visual Basic for Applications code to get a list of admitted patients and, perhaps, be updated as locations change.)

(g) (perhaps) defining higher level graphical objects that represent results or result trends, etc., so that healthcare application can offer them for embedding in compound documents.

Object-Oriented HL7, A Proposed Approach

We observe that requirements (a) - (e), above, are very similar to the requirements fulfilled by traditional HL7 messages. The data and event information that should be transferred among applications is essentially the same. Requirements (f) and (g), however, are related much more closely to the specifics of the complete object interface specification in use since they rely on the specifications of the complete object interface specification to determine the specific methods that must be offered.

It is possible to construct a fairly natural correspondence between HL7 messages and the conceptual invoking of an object (figure 1). We can divide the world of HL7 objects into two broad categories:

- *Healthcare application objects* that model the interface behavior of healthcare applications systems (e.g., ADT_Sender or Ancillary_ADT_Receiver)

- *Healthcare data objects* that model the subjects of concern in a healthcare application (patients, providers, encounters, tests, observations, etc.)

In this correspondence an HL7 message, then, becomes call to a method offered by a healthcare application object with parameters that are healthcare data objects. The specific methods offered by a healthcare application object map to the trigger events associated with current HL7 messages.

Figure 5 illustrates the admit_patient method.

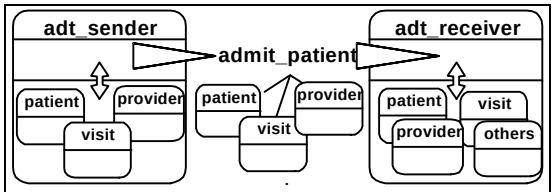

Figure 5, an admit_patient message.

An HL7 Abstract Object Model

If the HL7 group were to define a series of healthcare data objects, and healthcare application objects with their associated messages, we might call the product of that exercise an *HL7 Abstract Object Model.* By itself, the HL7 Abstract Object Model is not sufficient to use for communication, because it does not include the additional facilities of an implementation-specific complete object interface specification. We assert, however, that it would be a common subset of the definitions that would be required for developing specifications for requirements listed above. The HL7 Abstract Object Model should ultimately become a part of the HL7 Standard.

Much of the material that would go into preparing an HL7 Abstract Object Model can be found in existing resources. Trigger events and a starting list of attributes of Healthcare Data Objects can be found in version 2.2. The Joint Work Group for a Common Data Model will soon publish the metamodel and some preliminary notion of an object-oriented analysis of the entities that correspond to Healthcare Data Objects and their attributes. [5]

Special Interest Groups for Complete object interface specifications

Groups of users interested in actually communicating, however, will need more. They will need the specifications and software associated with a complete object interface specification. We would propose that special interest groups be formed within HL7 to produce complete specifications for implementing all the listed requirements. Their work product would represent a consensus of the subset of HL7 users interested in a specific technology and would be published in implementation guides. HL7 would not endorse any specific Complete object interface specification, but support groups of HL7 member that wished to gather around any one model.

Summary

Object-oriented interfacing based on the detailed specifications associated with complete object interface specifications holds the promise of providing plug and play interactions for cooperation among a wide variety of healthcare applications. While it is difficult to foresee all the requirements for such interactions, the approach described here will provide a conceptual basis for some of the most obvious requirements as a straight-forward extension to existing HL7 specifications.

[1]This table was compiled largely from material in Peter Wayner, "Objects on the March," Byte v 19 number 1, January 1994.

[2]Kraig Brockschmidt, *Inside OLE 2*, Microsoft Press, 1994, p. 191.

[3]Jean Bozman, "Skeptical users" *Computer World*, Feb 7, 1994, p. 95.

[4]Minutes of the Control-Query Group, January, 1994, HL7 Meeting.

[5]George Beeler, Abdul-Malik Shakier, et al., *Trial-Use Standard for Healthcare Data Interchange--Information Model Methods*,IEEE (in manuscript.)