Interoperability Issues in Large-Scale Distributed Object Systems

FRANK MANOLA

GTE Laboratories Incorporated, Waltham, Massachusetts. ⟨*fmo2@gte.com*⟩.

Issues of *interoperability* are receiving increasing attention as organizations move from mainframe-based "islands of automation" toward open, distributed computing environments, and as national efforts toward an "information superhighway" receive increasing attention. The demand for interoperability is driven by the accelerated construction of large-scale distributed systems for *operational use*. The Internet (particularly its commercial and research applications) is one such system, and much has been and is being written about it. However, in this paper we focus on enterprise-wide client/server systems being developed to support operational computing within large organizations to illustrate interoperability issues (Figure 1). Requirements for these systems are not speculative; numerous large businesses are building systems of this type today.

The architecture is distributed, and divided into three logical layers: applications, shared services, and data. The layers are only logical groupings; all components communicate via a common object-oriented messaging backplane. This reflects the increasing agreement that modeling a system as a distributed collection of objects provides the appropriate framework for integrating resources in these environments, and is illustrated by the number of standards activities that are moving toward adopting, or have already adopted, an object-oriented approach. System operations are based on explicit business rules representing business process definitions. The architecture supports integrated management of objects representing both business abstractions (customers, products), and elements of the enterprise and computing infrastructure (network elements, software, plant facilities). Applications are constructed from reusable components and users interact with the system via compound document and graphical interfaces (in many implementations, the "applications" layer is really just the graphical *interface* to the applications, with the application logic itself in the middle layer, integrated with or controlled by the business rules).

Such architectures require interoperability at many different levels, including the physical level (e.g., agreed-on data representations), and the object-model level (e.g., agreements on object interface characteristics). Much discussion of interoperability involves these levels. However, there are also requirements for interoperability at higher levels of abstraction. For example, there is a need for more complex agreements among object classes, such as those defining cooperation among the sets of object classes found in object *frameworks* [Gamma et al. 1995]. Interoperability requirements ultimately reach the level of *semantic* interoperability (agreements on *meaning*). For example, design of large-scale business systems are increasingly based on definitions of *business objects*, which attempt to reflect organization-wide agreements on the meanings of key busi-

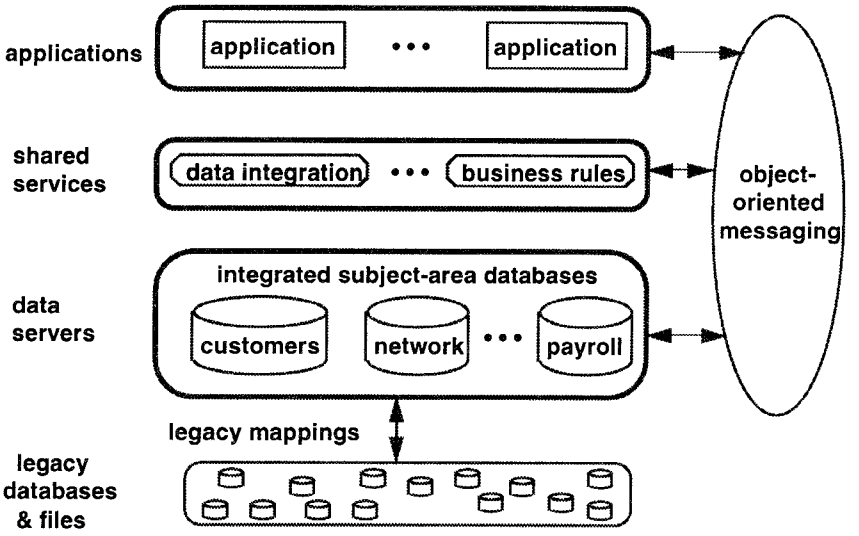

Figure 1. Enterprise information system architecture.

ness concepts and rules. Without agreements at this level, interoperability at lower levels is practically useless.

Providing increased interoperability in such architectures involves not only the increased use of standards, but also making more aspects of the architecture *explicit*, so that standards can be applied to them, for example, by documenting existing information (data and procedures), explicitly representing mappings between heterogeneous components, representing more aspects of the implementation explicitly as objects or object interfaces, and using explicit enterprise model and business process definitions. These approaches apply (in different ways) to all levels (from physical to semantic). The specifications being developed by the Object Management Group [1991; 1992; 1994], and such subgroups as its Business Object Management SIG, illustrate these approaches.

For example, a wide range of object models exists in different programming languages and application domains. The X3H7 standards committee has documented a number of these object models and ways in which they can differ [Manola 1995]. In order to achieve the required interoperability at the object-model level, these architectures usually adopt a common object model and define standard mappings between programming language models and that common model. The OMG IDL illustrates this approach [Object Management Group 1991]. At the implementation level, interoperability requires that the architecture support *object adaptor* components to provide interfaces to heterogeneous object implementations (including legacy systems), together with standards that assist in the interchange of object references, objects, and values. Agreements are also required on how *services*, such as transaction and query facilities, are provided for objects [Object Management Group 1994].

In order to deal with changes over its lifecycle, the architecture should also be *extensible*. This requires that key *internal* interfaces of the architecture, in addition to client-visible ones, must be made explicit. Ideally, the object model must also be extensible. This allows the addition of new object services, and new object model features supporting them, with minimum impact on existing components. One approach to providing this extensibility currently being explored is the use of *reflection* [Kiczales et al. 1991; Manola and Heiler 1994]. This involves making aspects of the architecture or

model (or its implementation) explicit as objects, so they can be altered easily or extended, rather than being implicit in the implementation. Reflection can also provide the basis for defining object model mappings, and provide hooks for extensions and interfaces to implementation components. The use of reflection is an extension of facilities found in a number of object models and implementations today. Manola [1993] surveys a number of reflective object models and describes the use of reflection in supporting various services in distributed object systems.

Reaching the numerous agreements that must be reached in providing interoperability in large-scale business systems can be a long process, as illustrated by the work of the OMG. Ideally, work must progress in parallel at multiple levels, from implementation to semantic, in order to reach the necessary agreements in a timely manner.

REFERENCES

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA.

KICZALES, G., DES RIVIERES, J., AND BOBROW, D. G. 1991. *The Art of the Metaobject Protocol*, MIT Press, Cambridge, MA

MANOLA, F. 1993. Metaobject protocol concepts for a 'RISC' object model, TR-0244-12-93-165, GTE Laboratories Incorporated. December (available from ftp.gte.com, directory pub/dom).

MANOLA, F., ED. 1995. X3H7 Object model features matrix. Doc. No. X3H7-93-007v10, February 14 (available from ftp.gte.com, directory pub/dom/x3h7; hypertext version at URL http://info.gte.com/ftp/doc/activities/x3h7.html/).

MANOLA, F., AND HEILER, S. 1994. An approach to interoperable object models. In *Distributed Object Management*, M. T. Ozsu, U. Dayal, and P Valduriez, Eds., Morgan Kaufmann, San Mateo, CA.

OBJECT MANAGEMENT GROUP. 1994. Common object services specification, Volume I, Revision 1.0, 1st Ed., OMG Doc. 94-1-1, March 1.

OBJECT MANAGEMENT GROUP. 1992. Object management architecture guide, Revision 2.0, 2nd Ed., OMG Doc. 92.11.1, Sept. 1.

OBJECT MANAGEMENT GROUP. 1991. The common object request broker: architecture and specification, OMG Doc. 91.12 1, Rev. 1.1.