Where Will Object Technology Drive Data Administration?

Arnon Rosenthal
The MITRE Corporation, Bedford, MA 01730, USA
arnie@mitre.org

Abstract

Several unifications that the application development process has long needed are now occurring, due to developments in object technologies and standards. These will (gradually) change the way data intensive applications are developed, reduce databases' prominence in this process, and change data administration's goals and participants. At the same time, the database community needs to ensure that its experiences are leveraged and its concerns are met within the new methodologies and toolsets. We discuss these issues, and illustrate how they apply to a portion of the Department of Defense (DOD). We also examine things that object technology won't accomplish, and identify research problems whose solution would enable further progress.

1. Introduction

Object technology will have evolutionary and revolutionary effects on data administration. Evolutionary changes include new descriptive constructs (e.g., user-defined datatypes) and new reuse mechanisms such as inheritance. This note focuses mostly on the more revolutionary changes that seem likely to emerge from the opportunities offered by Object Oriented Analysis and Design (*OOA&D*), object DBMSs, and multi-tier data architectures.

This section discusses several very favorable convergences that are occurring in industry. Section 2 discusses how the leading metaphor for application development will shift from databases to objects and types, and identifies data administration ideas that need to be added back to this mix. Section 3 uses DOD examples to illustrate some of the pragmatic situations that many large organizations may encounter in moving toward object technology.

Object formalisms provide an integrated way of describing data, functions (i.e., methods), and (sometimes) events. The same resources may be presented as a traversable web of objects, or as queryable collections. This integration permits representing "real world" concepts more naturally, and provides more natural modules of implementation, deployment, and reuse.

The maturing of object technology enables several other major convergences in the industry, which individually, and together, will have major consequences for data administration.

Convergence 1: A unified formalism across levels of abstraction.

- The same object formalism can express conceptual, implementation, and external schemas. The object query language can express derivations among them.
- It will be easier to give each community a schema customized to its needs, because a multi-tier schema architecture will be able to relate schemas within one formalism.

Two decades ago, ANSI/SPARC proposed a *multi-tier* schema architecture, that provided implementation, conceptual, and external schemas [Tsic77]. The idea was to give each community a schema that suits its world view. Though it became a staple of textbook advice, few organizations maintained the three tiers. Fatally, the conceptual tier typically used ER formalisms that possessed no widely-implemented query language. Developers could not express requests against the conceptual schema; therefore, they had no interest in it once the database was implemented. As schemas evolved, the conceptual schema was not maintained, and became useless.

There are several object formalisms that are suitable for being used at all levels of abstraction. They are typically rich enough to express schemas at all three tiers, and can provide both a (set-oriented) query language and a navigational (one-object-at-a-time) interface. In addition, they allow user-written methods that customize operations' behavior. Consequently, they remove the first set of barriers that frustrated multi-tier architectures. The problem is still difficult, though, as discussed in section 3.2.

Convergence 2: The design processes for databases and applications are interleaved, to form Object Oriented Analysis and Design (OOA&D).

- Applications will be seen as interacting with the object manager, rather than with the DBMS.
- Object modeling will often be done by developers who are not database experts.

The current design process consists of data design plus application design. The future split seems likely to be object design plus implementation design; the implementation then splits into data and code portions. Data administration's goals (including ability to share and integrate data) still apply, but the techniques will need to be adapted, and object designers will need to be convinced of their value. Techniques currently used within DOD include schemas, dictionaries and ontologies, schema integration, and use of declarative (SQL) mappings rather than black-box methods. From this list, only schemas are routinely used in OOA&D.

Convergence 3: OOA&D will use a common core of representations (UML and the Microsoft Repository), across vendors and lifecycle stages.

- Both vendors and user organizations will find it easier to assemble best-of-breed tools from multiple sources.
- Computer-aided Software Engineering (CASE) tool suites should improve rapidly, as market barriers drop, users need less training, and users benefit from tool synergies.

Until very recently, the representations used in software methods (and the supporting CASE industry) were hopelessly fragmented and incompatible. But while moving to OO-CASE, they have made enormous progress in standardization. Technical inputs came from many directions; the political push came from customers, the Object Management Group (*OMG*), and the Microsoft Repository, which (combining concrete form with market power) is being bundled with all Microsoft languages.

The result is that for many types of information involved in OOA&D, we have standard concepts, graphic notations and access interfaces. Leading OO-CASE vendors have agreed to export/import using the Unified Modeling Language (*UML*) [Rati98] and Microsoft's repository interface [Bern97]. UML will provide a standard core (also adopted by OMG) that vendors will interchange easily, as well as standard means of describing vendor-specific extensions. Also, because all UML diagrams are defined over the same metaschema, information can be seen consistently from the perspective of different lifecycle stages, e.g., class diagrams and use cases.

Currently, many DOD database designers use drawing tools (e.g., PowerPoint) for schemas. They reject database CASE systems as not worth the trouble and expense, for now. The future looks better. Standardization should reduce software costs and the learning curve. Improved integration should increase the payback, as the captured information is used for more purposes. As a consequence, there should be less tendency for design information to wither, unmaintained, as a system evolves.

2. Metaphor Shift, From DBMS to Object Management

For the past decade, application development tools (e.g., PowerBuilder) have seen information primarily as tables, accessed via a DBMS. We in the database community have become accustomed to being central to application development. It now appears that OOA&D (along with componentware [CACM97]) will become the default way of constructing new systems.

Below, section 2.1 examines the past and describes current trends in industry that appear to be moving toward object management as the primary metaphor for resources available to applications. Section 2.2 then discusses some of the consequent effects on data administration and on OOA&D, as the two combine.

2.1 Competition Among Metaphors

During the 1980s, application development environments saw available resources primarily as databases, described by relational implementation schemas and entity-relationship conceptual schemas. These descriptions could be exploited to generate access code and user interfaces (e.g., forms). Information could be captured incrementally and declaratively. Standardization was also important: Although SQL dialects and even catalog structures were not plug compatible, they were sufficiently similar that CASE vendors could wrench their tools to apply across multiple products.

Databases won over two other potential centerpieces. Transaction monitors (e.g., IBM's CICS) were the heart of 1970s systems, but these provided only an abstraction of the execution environment (e.g., multithreading, remote invocation), not of interfaces nor of the problem domain. Programming languages could describe functions as well as data, but were hopelessly heterogeneous in their data, processing, and storage models, and had no value-based query language.

Object formalisms, which offer even more powerful descriptive constructs (types and inheritance, methods), now provide a powerful rival. In the trade magazines and consortia, distributed object computing, as embodied in Java, CORBA, and DCOM, are considered critical to future systems. Databases get much less attention. The Object Management Group (OMG) has ~40 times as many members as the Object Data Management Group. OO interface formalisms have converged to a manageable number of competing (and cooperating) formal and *de facto* standards, e.g., CORBA, DCOM, and UML, and Java.

Equally important, object request brokers will be ubiquitous, encouraging applications to be written as invocations of objects' methods. A DCOM or CORBA request broker is now included with every Windows desktop and Netscape browser. In contrast, moves to make OO data management ubiquitous still face major barriers, including standardization, cost, resource requirements, and administration requirements.

2.2 Merging Data Administration Into OOA&D

If OOA&D becomes the dominant approach, it will need to include models that address the issues currently addressed by database-centered administration. This section summarizes the major changes administrators will see, and then examines how data administration can adapt to provide new functionality that OOA&D needs.

The biggest formalism change is that designers and implementers will see resources as individual objects and types, instead of seeing tables (i.e., sets) of typed tuples, or even tables of objects. The functions provided by DBMSs will continue to be crucial (notably collections, transactions, and a query language), but developers will often think of them as part of object management, not as a separate DBMS server.

The biggest usage change is that most developers will interact with *application schemas,* i.e., type definitions that match an application's world view. Both the conceptual and external tier of the three-schema architecture would be application schemas.

The biggest political change is that data experts no longer have exclusive turf, since OOA&D addresses data and function together. In fact, because producing methods is typically more labor intensive than producing data descriptions, method definition may receive greater attention (e.g., when selecting inheritance hierarchies). We now examine the changes in more detail.

First, the usual OOA&D scenarios will need extensions, to describe a data intensive application to the DBMS. Traditionally, data modeling is restricted to data that will be stored in databases, and table definitions that define both types (interfaces) and persistent collections of instances. OOA&D focuses on defining types. Hence one must specify, in addition, how objects are organized into collections, which objects (or parts of objects) are to be persistent, and what transaction model should apply to each method invocation. UML does not currently address these issues. OODBMSs provide the necessary mechanisms, but the interfaces and capabilities are often vendor-specific.

Second, at the enterprise level, it will be necessary to merge object schemas from multiple application domains. This task is familiar to database experts, but is less frequently addressed in OOA&D (or, for that matter, when combining text bases that possess overlapping kinds of metadata). Currently, the schema integration field can offer insights, research papers, and some limited industrial strength tools. The market for such tools should widen fast, as data warehouses and middleware simplify data access. The representation standards (convergence 3) will reduce vendors' costs and increase their market.

Administrators of schemas produced by OOA&D may have fewer decisions to make. Application schemas will be used as a system interface (unlike entity-relationship schemas), and hence be up to date. Compared with relational implementation schemas, they should change less often, have fewer oddities, and fewer redundant attributes (due to explicit inheritance and user-defined types). However, the formalism offers more options (e.g., ElapsedTime as an attributes or a method result).

Third, an enterprise needs a resource administration activity separate from individual applications. Object registries currently have different aims and coverage than registries of *data elements* (roughly, attribute definitions). For large enterprises, OOA&D should be complemented by a registry of data elements; this registry should address and exploit object technology, as discussed below.

2.2.1 Data Definition Registries and OOA&D

Data and object registries typically differ in granularity, and in their principal goals. OO component registries often aim at reducing development cost of an individual system, via reuse of fairly large granules. Data registries may have a complementary aim, to improve *interoperability,* i.e., the ability of separately-developed systems to work together. At an enterprise level, one cannot expect all systems to use the same set of type definitions; data element registries will still be needed in an OO world.

One form of interoperability concerns requests from one system to another. With object technology, these requests are method calls. Developers of various applications might get attribute and variable definitions from a registry, e.g., say, a textual definition of "arrival_time". The shared definitions will make it easier to use the concept in a method call. (Note that this approach avoids requiring that client and server use a consistent set of business objects.)

A second goal, more specific to *data*, is that shared definitions may permit organizations to reuse *instance* data, and thereby reduce redundant and inconsistent data capture. Achieving this goal reduces organizations' (human or sensor) operational costs, not just software development.

In DOD, individual organizations manage their own data and often see interoperability as an externally-imposed burden. The central data administration effort has interoperability as its prime mandate, and manages only data that appears in interfaces. They have registered ~15000 data elements and hundreds of schemas. Nevertheless, progress toward interoperability has been slow. Until recently (i.e., the WWW), it was too costly to provide inspection and definition tools to thousands of contractor and DOD sites, worldwide. Also, for organizations that have not previously shared information, short textual descriptions are an insufficient semantic basis for exchanging critical data. At a minimum, one must track who asserted the correspondence, and how certain they are [Rose97a].

Inheritance hierarchies and user-defined datatypes can reduce the labor of maintaining such registries. For example, much of an attribute's semantics might be captured in the datatype for its values. Currently, DOD has a centrally controlled list of domain datatypes (Time, Speed, etc.); in an OO environment, the list would naturally be extensible, so more semantics could be captured. Method arguments would also be included in this registry, since one system's attribute may be another's method result.

The OO community's view that a standard should permit tool interoperability would be a giant step forward for registry standardization. The ISO standard for registries [ISO97] contains much insight into registry maintenance, but less about how standards catalyze tool progress. Committee leaders have expressed disinterest in defining an invocable interface for the mandated registry functionality. Hence, unlike the OMG and Microsoft specifications, the ISO-11179 standard will not allow a user tool to work with multiple registries.

3. Pragmatics of Introducing Object Technology

This work is an outgrowth of efforts to advise the Department of Defense on data administration, particularly for planning and monitoring Air Force operations, and for central data administration. (Opinions are strictly the author's.) We use illustrations from DOD efforts (mostly tentative), but expect that the problems are shared by many organizations.

3.1 DOD Adoption of Object Technology for Data Intensive Applications

Recently. the Air Force's Chief Architects Office identified flexibility as its primary goal for future software. Technology is opening important new opportunities, e.g., for reducing personnel movement in a crisis by performing more tasks from bases in the US. The change will be gradual, and information systems must be able to change gradually. Perhaps more dramatic, unanticipated emergencies will be met by configuring customized task forces. Their information systems will need likewise to be customized, within a period of days rather than years. This requires that customization require a small enough effort to be done in-house, without going through the slow, costly government procurement process.

Object technology and componentware are widely seen as keys to flexibility. Today, even conservative organizations running RDBMSs want to add attributes that exploit new datatypes (e.g., image data blades) being sold by their current vendor. These organizations require administration of schemas that include such datatypes, and of the datatype definitions themselves. (Use of enhanced DBMSs as standard software components also requires guidelines, since the enhancement may potentially make the DBMS less stable).

Object database middleware (including object-relational products like UniSQL/M) are occasionally being used to provide integrated access to multiple data sources. As in many middleware-based systems, one needs separate schemas for the application interface and the implementation schema that encapsulates external systems. Administration of such multi-tier systems currently seems to rely on what the middleware vendor provides.

There are also groups developing new applications using OOA&D. They will want to store their information in object databases (whether native, or created by a persistence layer over a relational system). Currently, the tools seem to switch abruptly from class diagrams to database issues. Perhaps in the future, the transition will be less abrupt. These groups currently make limited use of the data registries.

There is widespread desire to simplify system administration, and this can slow the adoption of object databases. We know of one OO application that ran natively over an OODBMS, but which was rejected because owners of the deployment platforms did not want to administer a second DBMS. The application was accepted only after being rewritten (less elegantly) to employ already-deployed copies of Oracle. With similar motivation, Microsoft's repository hosts its object layer over existing engines (Jet/Access and SQL Server).

3.2 Why Objects Aren't a Panacea

Object technology eases some difficulties in building complex, flexible, systems, but many others remain, and complicate data administration. We discuss three of them below.

1. *Resources will not all be described in the same object formalism*. Heterogeneous formalisms require an organization to have more skills and more tools; also, derivations need to be decomposed into a query that derives the desired information, plus a "cast" between formalisms. The various industry sectors involved in implementing OO systems have each developed their own standard formalism(s): UML for the CASE community, CORBA and DCOM for distributed objects, and SQL3 and OQL for databases. Also, standards change, and each version is, in effect, a different formalism. DOD cannot upgrade tens of thousands of systems all on the same day (or year), so multiple versions will coexist.

Fortunately, the number of competing standards for each capability is fairly small, and there are moves to reduce inconsistencies. It may be feasible for a gateway to read an object's description in one formalism, and then present it in a different one. (When the gateway simply reveals an object through an alternate interface, it may be possible to ignore difficult issues such as inheritance mechanisms, creation and deletion services, and event models.)

2. *Application schemas will often remain distinct from implementation schemas.* The change of formalism does not remove the major drivers for organizing physical information: response time for critical tasks, encapsulation of external sources, etc. It seems best for implementations to be built from first class objects, and that the mappings between application and implementation schemas use formally-specified, non-proprietary languages.

3. *Multi-Tier schemas are needed, but are difficult to support*. OOA&D promises to present diverse developers and users with objects that match their mental concepts. Since the physical database conforms to a single implementation schema, one must have additional tiers of interfaces until one provides views that match application concepts, as seen by each user community. We call a structure with two or more schemas, related by derivation, a *multi-tier schema.*

Ideally, these application views would have a full set of generic methods. To access data, one would start with GetAttribute and Query. Beyond these, one wants generic methods for Update, DefineTrigger, GrantPermission, and many other tasks. Each of the above methods must be implemented in terms of methods of objects at the implementation tier. Worse, as well known from view update, the mappings tend to be ambiguous, so a data administrator needs to select which approach is suitable for each attribute being mapped [Rose97b]. The scale is daunting, both for tool vendors (who must implement the mapping and decision tools for each generic method) and for administrators, who must make and maintain the choices.

Object technology does not address the above difficulties directly, but does ameliorate them. Type definitions and constraints require fewer circumlocutions than in relational schemas. Inheritance and user-defined types aid in reuse. And the customized semantics, once chosen, can be implemented as methods. However, developers' tendency to implement derivations in C++ or Java need to be resisted. To permit end-to-end query optimization and to offer hope of generating the other methods (e.g., view updates), the derivations must use a theory-friendly query language (e.g., OQL, SQL3).

4. Final Comments

In some organizations, data administration groups have failed to help system developers, and have therefore been dismantled [Nath97].

The trends discussed here provide new challenges, but also new opportunities for data administration to be effective. We end by posing some open issues:

- We cannot predict which standards will win, nor exactly how the technologies described here will be embodied in industrial-strength products. What can an application organization do now, preferably with short term payoffs, to be better prepared for the future?
- We need to improve or develop theories and tools that map ancillary operations (e.g., define trigger, grant privilege) between tiers. Also, design tools that operate in the large (e.g., to select an implementation for an application schema) create unmanageably large changes when a system evolves. How can slight changes at one tier be mapped to similarly small changes at tiers above or below?
- How do the trends explored here (e.g., multi-tier schemas) play against other trends, such as OO application frameworks, and design patterns?

References

[CACM97] "Object-Oriented Application Frameworks", *Communications of the ACM*, special section, Oct. 1997.

[Bern97] P. Bernstein et. al., "The Microsoft Repository", *Very Large Database Conf.,* Athens, Greece, 1997. For further information, see http://www.microsoft.com/repository/

[ISO97] International Standards Organiation (Gilman ed.), "ISO/IEC 11179 - Specification and Standardization of Data Elements", www.lbl.gov/~olken/X3L8/drafts/draft.docs.html

[Nath97] M. Nath, personal communication, describing cuts at his state's Department of Natural Resources.

[Rati98] UML Resource Page, Rational Software, http://www.rational.com/uml/index.html

[Ros97a] A. Rosenthal, E. Sciore, S. Renner, "Toward Integrated Metadata for the Department of Defense", *IEEE Metadata Workshop*, Silver Spring, MD, 1997. www.llnl.gov/liv_comp/metadata

[Rose97b] A. Rosenthal, P. Dell, "Propagating Integrity Information in Multi-Tiered Database Systems", *Workshop on Information Quality*, Cambridge, MA, 1997.

[Tsic77] D. Tsichritzis, F. Lochovsky, *Data Base Management Systems*, Academic Press, 1977.

Acknowledgement

The author received generous help from Leonard Seligman, Sandra Heiler, Scott Ambler, and Chris Clifton.