K/CSD/INF-92/27

MARTIN MARIETTA

OBJECT TECHNOLOGY
A White Paper

Sara R. Jordan
Lloyd F. Arrowood
William D. Cain
Wesley M. Stephens
Barry D. Vickers

Computing and Telecommunications Division
and
Engineering Division (Cain)

Martin Marietta Energy Systems, Inc.
Oak Ridge, Tennessee

May 11, 1992

Prepared by the
Oak Ridge K-25 Site
Oak Ridge, Tennessee 37831
managed by
Martin Marietta Energy Systems, Inc.
for the
U. S. DEPARTMENT OF ENERGY
under contract DE-AC05-84OR21400

OPERATED BY
MARTIN MARIETTA ENERGY SYSTEMS, INC.
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

K/CSD/INF--92/27

DE93 006037

Foreword

Object-oriented technology has gained considerable attention in information systems, CAD/CAM, and software engineering publications in recent years. This white paper presents the fundamentals of objects, their impact on software development, plus the emergence of object-oriented data bases and standards.

In addition, this assessment looks at the various OOT activities on-going in Oak Ridge, Kansas City, Savannah River, Los Alamos, Albuquerque, and other DOE sites. Recommendations and references are presented to the managers and programmers who may adapt object-oriented software.

Acknowledgements

Many people have contributed information, insights, and lessons from experience to help us produce this white paper. Special thanks are given to the Martin Marietta reviewers of an earlier version of the paper: David Dieterich of the Computer Aided Productivity Office in Bethesda, and in Denver, Barry Bounds, Tom Couchman, Don Herkimer, Bob Owens, Don Rudisill, and Randy Stafford. In Oak Ridge, the advice and assistance of Spivey Douglass, Phil Jones, Bill Kimmerly, Jim Snyder, Jerry Sullivan, and Doyle Turner are much appreciated.

More than fifty people across the DOE complex have been interviewed to gather information about the experience already gained using object technology. Some of these people have also contributed appendices to this paper which describe that experience.

Permission to use several figures and tables from *Intelligent Systems Strategies* and *Object-Oriented Strategies* was graciously granted by Curtis Hall and Paul Harmon of Cutter Information Corporation. Use of another figure from *Information Week* was granted by CMP Publications. Additional assistance (and use of some material) was provided by the ANSI Object-Oriented Database Task Group, specifically Elizabeth Fong and Craig Thompson.

We greatly appreciate the excellent support given by Tom Willoughby, Mary Schulte, and Louise Egner in the final stages of producing the document. Finally, we wish to thank the Energy Systems managers, Chuck Hall, Ken Sommerfeld, Ron Leinius, and Bob Henderson, for their continuing support through this technology assessment.

OBJECT TECHNOLOGY

Table of Contents **Page(s)**

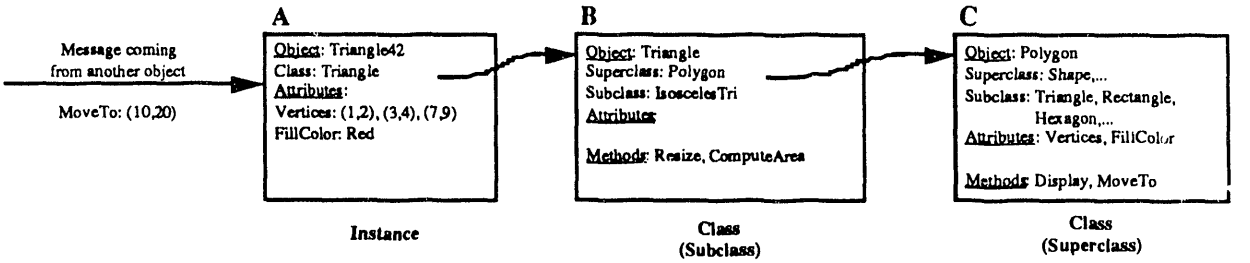
Executive Summary

Object-Oriented Technology (OOT), although not a new paradigm, has recently been prominently featured in the trade press and even general business publications. Indeed, the promises of object technology are alluring: the ability to handle complex design and engineering information through the full manufacturing production life cycle or to manipulate multimedia information, and the ability to improve programmer productivity in creating and maintaining high quality software.

Groups at a number of the DOE facilities have been exploring the use of object technology for engineering, business, and other applications. In this white paper, the technology is explored thoroughly and compared with previous means of developing software and storing databases of information. Several specific projects within the DOE Complex are described, and the state of the commercial marketplace is indicated.

What is object-oriented technology?

Object-oriented technology promotes a paradigm in which an application domain is analyzed to identify its essential entities, their important attributes and relationships to each other, and their behaviors. The entities are the *objects*, each one representing some concrete thing or an abstract notion relevant to the problem domain. Objects are organized into *classes* of objects which describe common attributes and behaviors. A class may have subclasses which specialize or refine the abstraction represented by the class, and which *inherit* the state and behavior of the class and its superclasses. When an *instance* of a class is created, the instance assumes the default attributes and behavior of its class (or type). The behavior of a class is a set of operations that its instances can perform to produce a result, or actions that they can cause to happen; each such operation is called a *method*. An essential feature of objects is that they are *encapsulated* so that direct access from outside to the values stored in their attributes is impossible; the only access to an object or its actions is by sending a *message* to activate a selected method. These concepts are illustrated in the figure showing three objects below. In the figure, instance A will first take its attributes and methods from object B; specific attribute values are stored locally in A. If A needs information not stored locally or in B, B will inherit the information from object class C (or C's ancestors, if they exist). Some of these terms may be familiar to groups who have used Ada, which is an object-based programming language promoted by the Department of Defense.



Use of objects to model an application domain yields a new approach to software development. Instead of having files or tables of data on which procedures operate, an object-oriented application consists of a collection of objects, causing desired behaviors and actions by sending each other appropriate messages.

There are several components in object-oriented technology. The earliest part which appeared was **object-oriented programming (OOP)**. The most famous object-oriented programming

languages are Smalltalk (the oldest one) and C++; the object-based language Ada is also well-known. When large organizations seriously began to explore software development using the object model, then the terms **object-oriented analysis (OOA)** and **object-oriented design (OOD)** were coined. Well-known OOA authors are Coad & Yourdon, Shlaer & Mellor, and Wirfs-Brock. Prominent OOD authors are Booch, Coad & Yourdon, and Rumbaugh. Finally, the successful development of object-oriented applications led to the need for **object-oriented databases (OODB)** such as Object Design's ObjectStore, Servio's GemStone, and Versant Object Technologies' Versant.

Object-Oriented Databases

In order to fully exploit the ability of objects to model complex information, we need some means of storing objects over time. Use of conventional databases (such as relational) requires collapsing these structures and their relationships into flat tables of information, and rebuilding the structures whenever they are needed. This causes a significant performance impact. Object-oriented databases are being developed to provide more direct representation of objects, including behavior in some products. Our activity in advanced engineering and manufacturing applications is cause for keen interest in this area of the technology.

The development of OODBMS has proceeded in two directions. In one, object-oriented programming languages have been augmented with many of the services provided by commercial database systems, e.g., persistent storage of objects, archiving facilities, transaction management, and query languages. Such OODBMS, termed "language-centric", are predicated on the belief that it is easier to implement database capabilities into an object programming language than to augment type capabilities in existing database systems.

The second common approach to OODBMS development has been to extend or evolve existing relational database management technology to include support for additional data types and query language extensions to accommodate objects. The extended data types for these "database-centric", or extensible, database management systems may include text, graphics, voice, image, and possibly also procedural data.

Of the two approaches to OODBMS development, most of the current marketplace OODBMS offerings have language-centric origins. Some of these language-centric systems provide excellent performance on applications whose data require the object representation. While there are some database-centric OODBMS products now in the marketplace and even more soon to become available, at present they are still fewer in number and also generally offer lesser degrees of object support "richness" than do the competing language-centric products. Database-centric systems should theoretically lend themselves well to an evolutionary integration path with existing relational database applications. Proponents of database-centric systems point out that there are no new coexistence or communication issues with this approach, but rather graceful extensibility and migration of existing applications is attainable. They also state that these OODBMS products are "more production quality", which is a commonplace requirement of traditional database applications.

What are the benefits of OOT?

Object technology gives improved *ability to represent complex and heterogeneous information* and to manipulate it, for example in applications requiring engineering/manufacturing information, multimedia information, or graphical user interfaces. Because of the way this information is modeled and represented in encapsulated form, it encourages *reusability* of software components,

both through the inheritance of common attributes and behavior to a more specialized object, and through the sharing of **libraries** of these object classes across applications and organizations. The *reliability and testing* of object systems is improved because the units can be tested out thoroughly on creation, and additional retesting is minimized when the units are reused elsewhere in different contexts. *Maintainability* is enhanced by the localization of data and procedures, and enables program updates to be made efficiently since they need to be made in only one place. *Extensibility* is provided through customizing existing objects with more specialized attributes or behaviors to fit a specific application, and enables extensions to the program which are variations of the existing system to be made with minimal effort and impact on previously written code.

An organization's collection of object class libraries can be collected into a *repository* which serves as a reusable, multilingual software base. Development of new applications becomes a process of finding the appropriate classes which already exist, and extending or customizing them to provide the behaviors needed for the new situation. This mode of software development is generally faster, cheaper and better than traditional structured analysis and design methods of software production.

What are the costs and impacts of OOT?

For maximum benefit from the technology, a fundamental change to our software development processes is necessary, though changing to it can be gradual. The biggest challenge of adopting OOT is getting a body of qualified object-oriented software engineers. Both **education** in the object paradigm, with a resulting shift of mindset, and **training** in the needed tools and methods will be required.

All software developers must be familiar with the corporate information repository and the libraries of objects (class libraries) available for their use as components in constructing new applications. It is wise to use consultants in the early stages to act as mentors and teachers for the people learning the technology. There is often a long period of analysis, prototyping and design before production-ready software appears. For all these reasons, there will be an initial period of investment of staff time and resources for any organization to acquire the needed base of experience and knowledge of available software components. After that period, realization of the benefits of object technology will accelerate because providing new applications becomes a matter of extending the software base to provide the needed increment of capability.

Creation of a central object library or repository function will require additional resources and management leadership. This will be the cost to build, maintain, and provide reuse mechanisms for the organization's class libraries.

OOT Activities Within the DOE Complex

Object technology has been applied within the Complex for several years in a number of application areas. In some cases objects have been required to represent the inherent complexity of the problem domain. Thus objects are being used in a couple of projects to model nuclear reactors and in several projects to model product definition through the design-to-manufacturing cycle. The quality of multimedia interfaces which can be provided with an object approach has aided production of multimedia information systems and expert training systems for maintenance of complex machinery.

Recommendations for Using OOT

As object technology consultant Jon Hopkins says, OOT is a **revolutionary** technology, but adopting it must be an **evolutionary** process. Here are the necessary steps which we recommend in Chapter 8 to realize the benefits of object-oriented technology.

Educate those in the organization who develop and use software about object technology and its potential impact on software development. Sell the object approach to all concerned:

- **senior managers** who will be needed for money and support, and who can serve as champions in times of waning resources
- **middle managers** who must fit object-oriented projects in with the rest of the budget, schedule, and political realities
- **technical people**, who may not be uniformly receptive but whose skills and commitment are essential to success.

Make sure that all these groups have **realistic expectations**.

For those sites which have not already done so, use **pilot projects** to gain experience. Formulate a methodology which adapts the object technology to local needs. Make and follow a carefully thought out **implementation plan**. Be sure the major objectives are known, along with the metrics which will be used to evaluate each project. The success or failure of a project has more to do with adequate planning, commitment, and expectations than with the choice of a particular technology.

Work with some of the **best people first**, not the ones who can be spared.

Use **consultants** to assist in education, mentoring, and assisting with pilot projects.

Keep up-to-date on the **vendors and standards groups**.

Maintain **good communication among all involved** in working with object technology.

Begin building a **software reuse library** for the organization. By 1993, each site should set up a small group to: 1) establish naming conventions for the objects used to model the organization and 2) begin building the organization's reuse library. The reuse group should coordinate closely with any information engineering efforts already under way in the organization.

Why should we go through this extended process to evaluate and adopt object oriented technology? Because **reusable software components are *strategic assets*; they have great value to the organization as a repository of our collected expertise and a factor to differentiate us from our competition. This realization attaches great importance to proper management of these assets and any organizational changes necessary to implement their use.**

Chapter 1:

Introduction

Several groups at sites around the DOE Complex have been separately investigating the potential of object-oriented technology (interchangeably called object technology) for providing better solutions to their problems and for improving the software they produce and use for their respective tasks. The motivation for these efforts lies in the way each group's problem can be perceived and solved. The object model allows users and developers to think about their requirements in terms of the way the problem domain works, not in terms of the way computer structures work. In many cases the object approach of identifying objects and their behavior suggests a solution to the problem at hand which is much more natural than that provided by conventional information technology approaches.

Engineering groups are evaluating object databases because conventional databases have proved inadequate for representing and performing with complex component design and manufacturing information. Database and Computer Integrated Enterprise people are interested in the added capabilities that object technology will give them to handle the organizational information that drives our enterprise. Knowledge-based systems people have long recognized the power of frames and (later) objects as a means of representing and reasoning about complex relationships and behavior of the entities in their application domains, and have followed closely the maturing capabilities of object systems and tools.

An Energy Systems team representing all these interested groups has conducted a technology assessment, and this white paper summarizes the conclusions of that study. We have canvassed the Energy Systems and other DOE Complex groups who have been working with object systems, and have surveyed relevant literature and vendor offerings. We hope that the information presented here will allow all the groups to benefit from what has already been learned, will identify existing projects and contact points within the various site organizations to allow sharing of experience, and will reduce the amount of search necessary to identify the best approaches to use on future software efforts. Object technology is receiving tremendous interest and attention from many who benefit from computing, including funding agencies as well as internal organizations in need of automated solutions. More effective use of object technology will enable us to provide high quality competitive solutions to a wide range of problems, some of which in the past have proved to be unwieldy.

In this chapter we describe the need for object-oriented technology and give a brief overview of the technology and its benefits, risks, and history. In later chapters, different aspects of the technology are defined and discussed in much greater detail. The final chapters assess how the technology is progressing and what our team thinks should be done to take advantage of it. The Appendices include a glossary of terms used, descriptions of leading object-oriented programming languages, and recommended readings. The final appendices are descriptions of project experiences from around the various DOE sites, with information for contacting project representatives.

1.1. Motivation: the Current Software Crisis

For a number of years we have acknowledged that there is a software crisis, that the ability to produce software has grown at only a small fraction of the growth rate and cost reductions of hardware and telecommunications, and the demand for software is far greater than the supply. In some places, the software backlog is measured in years. The situation is that "software is too costly, of insufficient quality, and its development nearly impossible to manage." [COX90b, p. 26] Moreover, applications continue to grow larger and more complex. Today it is not uncommon for large application systems to have millions of lines of code, as opposed to the few hundred thousand lines for a large system several years ago. Finally, much current software effort is on maintenance, making modifications because of changes in user requirements, data formats, bug fixes, and so on. Much too often applications are developed as new efforts, and often the same functional code is written again and again by the same or different people over time or different projects.

1.2. The Software Industrial Revolution

In recent articles, industry writer Brad Cox of Stepstone Corporation has made a compelling case that we must have a "software industrial revolution". After the great Industrial Revolution, the Age of Manufacturing has matured and is now being redefined by advances in technology. The phenomenal achievements that manufacturing brought to computing, communication and transportation have created a new Information Age. We now have mountains of data, mostly irrelevant and useless at any given time, so we must have ways to filter and refine that data to yield useful information. The central strategic resource has become **software.**

The software crisis has created a "vast economic incentive" that will continue extensive growth as our global economy moves into the Information Age. Cox argues that the software industrial revolution will come whether our programmers want it or not, because software **consumers** will ultimately determine the outcome. The consumers have the most to gain, and will control their destiny with that time-honored behavioral modification tool, **money**. [COX90a] If we wish to be competitive and hold our position as providers of software solutions, we must seek improvements in our methods.

The hand-crafted, "start from scratch" attitude of conventional software development has become hopelessly archaic for a maturing software industry. Many would argue that the key element to the coming software industrial revolution is changing to a product-centered, object-oriented view of software. That is, we must create a reusable standard parts marketplace, where problem specialists can procure low-level, pluggable software components to assemble higher-level solutions.

1.3. Definition of Object-Oriented Technology

The term Object-Oriented Technology (OOT) is used to refer to the approach of solving problems by understanding objects and their behavior within the context of a problem domain. That is, objects combine appropriate data structures and operations into integrated components for use in software systems. OOT gives a different way of looking at solving problems with software. In conventional systems a program is composed of algorithms plus the data structures on which they operate, or it is built by parallel and separate development of entity-relationship models and data-

flow diagrams. For large or complex problems the resulting need to integrate data and procedure becomes very difficult. With an object approach, the objects or entities (usually high level) of the subject domain are identified. The attributes of those objects are described and modeled with data structures *encapsulated* with their appropriate operations (called *methods*). Because the objects are encapsulated, internal details of implementation are hidden and inaccessible, but users of an object need only know the methods to invoke in order to retrieve or change a value or cause some other action. A program consists of a collection of these objects, sending each other *messages* to activate the desired behaviors. A rapidly generated prototype can be very advantageously used here to test the solution and provide feedback for changes. The power of the object approach is that the problem can be attacked in pieces and integrated easily. It is easy to get a portion of the problem solved and then add more complexity incrementally. A report of the American National Standards Institute Object-Oriented DataBase Task Group describes it well [THOMP91, p. 1]:

> "In programming, the use of objects has emerged as an important approach to making better data modeling support available to programs while making programs less aware of and less dependent on details of how data structures and operations are implemented."

1.4. Handling Complexity

It seems that no software organization ever has the resources to produce all the software that is needed, and the backlog keeps growing even when people are added. In addition, there is increasing demand for software which manages complex and highly interrelated data types (and combinations of them). Examples of these are found in the areas of computer integrated manufacturing or in multimedia information systems. Another pressure is the inexorable push toward highly interactive and graphical interfaces to software. Software consumers accustomed to Windows and Macintoshes want such interfaces, and some computing leaders estimate that about seventy-five percent or more of a highly interactive system's development effort is directed at the user interface.

Creating such software systems strains the capabilities of conventional software approaches, but fortunately, object technology with its data encapsulation, data abstraction, and "black box" capability for hiding details has shown itself quite valuable for developing these complex environments. Object-oriented databases show great promise for dealing with complex data types, and these will be discussed in Chapter 5. The application of OOT to user interfaces and as front ends to other programs allows designers to create impressive and flexible interfaces with minimal effort. Thus it allows the designers to focus more attention on the underlying algorithmic and data management problems of the application.

1.5. Reusability

A much touted benefit of object technology is **reusability** of software modules, achieved via mechanisms such as the inheritance of common attributes and behavior through a class hierarchy, or libraries of encapsulated class modules available for use in many applications. General-purpose software components can be developed for repeated use across a project, an organization, or even as standards for use across the country. With a large body of reusable code available to developers, much of the effort and cost of new applications disappears. However, reuse does not happen automatically. Software organizations must plan for it, reward developers for reusing software, and design their software for reuse. The experience of AT&T's Bell Labs with object-

oriented programming illustrates the type of gains achievable. Over a three year period, they moved new development from C to the object-oriented language C++. When using C, they had averaged 80% new code on a given application, with 20% of the code coming from existing libraries. After the three year transition to C++, their analysis showed only 20% of the code had to be written for an average application; 80% came from existing libraries of code. That meant a four-to-one reduction in the amount of new code to implement a new application. After taking account of all the software life cycle, the total development time was often decreased by 60%. [ATWO89] See Sections 2.5.2 and 4.5 for further discussion related to this topic.

1.6. Software Quality

It is widely agreed that software reliability and modifiability are enhanced if there are no dependencies between different software modules in a program. For each module, the variables used should exist locally by default, and any shared data or passed values should be explicitly declared. Object-oriented programming languages (OOPLs) have features that promote efficient development of such modular, minimally coupled systems.

OOPLs promote the definition of classes of objects, bundling or encapsulating their data attributes along with their functional behavior. Data and behavior can be inherited from a parent class and can thus be reused. If a new subclass needs additional attributes or new behavior, the parent class can be specialized or customized to efficiently produce the new subclass. Each class can be thoroughly tested at creation and can thereafter be reused, generally without concern for its internal correctness. Classes have a well-defined interface (no direct access from outside is possible), so building a new application requires identifying the appropriate classes in the library and tying them together, along with any needed customization. When a class requires internal revision, it can be carried out without external impact as long as the defined interface and behavior are maintained. A more detailed description of the characteristics of object-oriented programming is included in Chapter 2, or see Appendix A for definition of terms.

Summarizing the benefits, the use of object-oriented programming will greatly ease the modification and repair of large programs, and will provide an additional and powerful way in which programmers may analyze and solve their application problems. Taking advantage of reusability, programmers will increasingly build new programs from libraries of tested components, and thus will be able to more quickly create large, powerful and useful applications. This should go far in helping to meet the software crisis.

1.7. Risks and Costs

The goals of high productivity and reusability can be difficult to achieve. Programmers must be carefully trained, not just in the syntax of object languages, but in fundamental changes to the programming model to produce very general classes designed for wide use. They must be familiar with available class libraries and know how to build applications using them. Successful reuse requires adequate archiving and repository technology, plus procedures for using this technology which are beyond any in common practice. Adopting an object-oriented programming language without investing in libraries and the procedures for using them will not significantly improve either reusability or maintainability of software. On the other hand, there is a definite level of risk in making any significant investment in tools, training, and commitment to specific vendors in the absence of mature standards and a rich set of class libraries and other tools to support the full

software life cycle. These risks and costs, and the accompanying cultural issues that arise, will be further discussed in Chapter 4.

1.8. Haven't I Heard This Somewhere Before?

Some who read this description of current software problems and motivation for object-oriented technology, particularly the parts about software reuse, may recall that very similar arguments were made for using the programming language Ada. The hoped-for benefits from using Ada have been only partially achieved, however. Some of the reasons may be technical. While the Ada language supports data abstraction and information hiding, it does not provide inheritance, dynamic binding or polymorphism (more on these topics in Chapter 2). Given the lack of these features, Ada does not yield the benefits of reuse as readily as pure object languages like Smalltalk. There are now toolsets available (such as Classic Ada) which are an extension of the Ada language and provide many of the object-oriented capabilities of languages such as C++ and Smalltalk.

Some companies have had impressive successes using Ada, but not all. Whether or not Ada offered all the features of a true object-oriented programming language, some observers feel that the problem with early Ada use was "Adatran". Instead of changing the whole paradigm or approach to software, they feel that programmers were still allowed to write Fortran-type code and run it through the Ada compiler, which of course did not result in real object-oriented systems.

Successful use of Ada or any true object technology will require significant organizational commitment (from top management down through the software developers) and education. There should be pilot studies, and core teams should follow object technology closely. However, the technology will be incorporated into widespread organizational use only when its processes and tools become mature enough to support a production environment with adequate methodology. At that point, all involved should be trained and motivated by management (mandated, if necessary) to take ownership of the tool or procedure. Perhaps much of the difficulty experienced with Ada did stem from the immaturity of the technology and the lack of support mechanisms and CASE tools. However, as Ruben Prieto-Diaz of the Software Productivity Consortium points out,

> "The problem is not lack of technology but unwillingness to address the most important issues influencing software reuse: managerial, economic, legal, cultural, and social. On the one hand we have our technical toolbox full but on the other, we cannot use these tools effectively because a proper infrastructure is absent." [PRIE91]

At any rate, for long-term success with object technology, we will have to learn from the Ada experience.

1.9. A Brief History

Although object-oriented technology has received a tremendous amount of coverage in the press for the last few years, it is certainly not a recent development. The first object-oriented programming language (OOPL) was Simula, a hybrid which grew out of Algol in the 1960's. Smalltalk was the first pure OOPL; everything, even an integer, is an object, and the language provides a complete programming environment. Combining features from both Simula and the artificial intelligence language LISP, Smalltalk was developed by Alan Kay with others (including Adele Goldberg) at the Xerox Palo Alto Research Center (PARC) in the 1970's. The 1980 version

of the language (Smalltalk-80) has been the most widely used pure OOPL. Today, the most widely used object-oriented language is C++, which was developed as a hybrid from the C language in the early 1980's. We will discuss both C++ and Smalltalk more in later sections of this paper.

Artificial intelligence workers must represent complex real world knowledge in order to enable intelligent behavior from their systems, so object-related notions were developed in a number of languages and systems. Representation of knowledge in frames (closely related to objects) was seen in the 1970's, and Lisp-based object-oriented languages were developed at Xerox (LOOPS) and MIT (Flavors). The first commercial expert system shell KEE (Knowledge Engineering Environment) was introduced in 1983, and featured object techniques as well as rule-based inference.

Perhaps a more compelling drive toward the use of object representations comes from engineering and manufacturing, where there is a need to represent complex product information throughout its entire life cycle. Existing database and applications technology have not been adequate to provide the needed software solutions, so these groups have been among the first in industry to explore the possibilities of OOT, specifically object-oriented database management systems.

One last bit of history might be interesting. In the late 1970's, Steven Jobs of Apple Computers visited Xerox PARC and was quite impressed with the wonderful interfaces that were being developed there using Smalltalk. He hired away several of their people and began development of the Lisa and then the enormously successful Macintosh computer. To help spawn software for the Mac, Apple created the MacApp object-oriented development environment. Later, the development of the more limited but very friendly object-centered Hypercard environment, has allowed people with little programming skill to create interesting, highly graphical applications.

Chapter 2:

Object-Oriented Software Development Why and What It Is

2.1. Traditional Software Development and Its Problems

2.1.1. Structured Analysis and Design

Most current software development is based on structured methodologies which were originally formalized ten to fifteen years ago. These traditional approaches, such as those espoused by Yourdon, Constantine and DeMarco concentrate upon a top-down view of the system and the functions it is expected to perform. The concept is to structure programs by decomposing the initial task into a number of separate functions and then applying functional decomposition to each smaller function. The decomposition continues until a level is reached which can be converted into programming code. Data flow diagrams (DFDs) depict the data as they are passed from one function to another. During the 1980's, structured methodologies were expanded to the full software life cycle, to regulate and prescribe what to do in the analysis, design, implementation, and testing of the software system and how to move it into production use and maintenance. Even so, software methodology authors such as Ed Yourdon and Bertrand Meyer [YOUR90; MEYE88] now see several major problems with this approach:

1. Structured analysis and design methodologies place **too great an emphasis on modeling function, and too little importance on modeling data.** Data modeling is now seen to be at the core of software development, as will be discussed in the following sections.

2. The structured methodology, with its waterfall model representing a single path through the development stages, **fails to recognize the evolutionary nature of software development.** This failure to embrace iterative development and prototyping has long been a problem for software organizations in the real world, where software requirements evolve as the customers and developers learn more about their domain areas and see how initial software efforts operate on those areas. Rumbaugh et al. [RUMB91] note that the structured, procedure-driven approach is more sensitive than a data-driven approach to changes in the requirements definition, since requirements generally involve system functionality rather than the underlying data structures which model the problem domain.

3. The structured analysis and design diagram notation gives **little or no mechanism for emphasizing reusable components.** Each new project is approached as a new intellectual exercise, starting afresh to build each new application. Even within a single project, the top-down analysis of needed system function inhibits reuse. As the hierarchy forms, the same function may be replicated in several of a system's functional subtrees, perhaps by different teams of developers. Failure to recognize this redundancy adds additional effort to the project and makes verification (elimination of bugs) harder.

4. Structured methods provide **little help with the user interface.** Back in the 1970's when the structured methods were developed, user interfaces were just streams of characters, but today's graphic user interfaces (GUIs) are quite different, often resulting in Window, Icon,

Menu, Pointer (WIMP) systems. Indeed, Bill Joy of Sun Microsystems estimates that up to seventy-five percent of the logic of current WIMP systems is associated with the user interface. Object technology gives a much more effective and productive way to implement such systems.

2.1.2. Data Driven Software Development

During the Eighties, increasing recognition of the importance and complexity of the data structures needed by programs led many groups to the adoption of a data-driven approach. For this approach, a corporate data model, usually being an entity relationship (ER) model is developed in addition to the procedural component. In ER information modeling, an entity is a person, place, concept, or thing about which an organization wants to store information. An entity-relationship diagram shows entity types in a corporation and the relationships between them. However, as Coad and Yourdon state, "Information modeling is a partial method", for ER models lack the behavioral characteristics of the object-oriented paradigm [COAD91a, p. 28].

In today's modern CASE (Computer Aided Systems Engineering) environments using *information engineering* or information resource management, corporate data models are constructed in parallel with the procedural component. The business rules of the organization may be partly represented in the ER model (via data constraints and cardinality) but reside mostly in the procedural component of the system. As the top-down analysis on both sides progresses, great efforts are needed to assure that the two components are compatible. The integration of the data models into the structured procedural decomposition has been difficult to achieve, however.

Let us see how the data-driven approach addresses the problems of the structured approach listed above.

1. The problem of too little emphasis on modeling data is addressed to some extent by the evolution to the information engineering approach. With the object-oriented approach to software, however, the data structures, their relations, and the functions they perform are packaged and implemented together as objects. This yields an even closer fit to modeling the real world of the application domain.

2. The use of a static requirements specification and its failure to evolve are somewhat alleviated in modern information engineering practices such as joint (or rapid) application development (JAD, RAD) sessions which gather computing professionals and customers from the application domain area for intensive group work, especially in the initial stages of a project. This interactive process, coupled with prototyping tools to show customers the results of their efforts quickly, compresses much of the learning process into several weeks instead of many months with the older process, and yields better requirements for concluding the development of the system. With the dual development of data and procedure, however, changes to either side of the system must be scrupulously traced through the other side of the model to detect all the needed modifications. With the encapsulation provided by the object approach, the impact of such evolutionary changes is minimized.

3. An organization's commitment to use the information engineering approach does enhance reuse of analysis in certain ways, because the corporate data model is developed cumulatively over time and can be used repeatedly in future applications of the organization. Object-oriented technology promotes the reuse of software, or "design by extension", by providing class

libraries of useful, general software components and inviting their reuse, either directly or by inheritance of their attributes and methods into new subclasses specialized for the new application. In addition to software modules, the objects reused might be more abstract components of the software life cycle, from analysis fragments to test suites.

4. Use of the data-driven approach does little to change the user interface situation. Even today, most CASE environments are still generating menu screens using Seventies technology.

2.2. The Object-Oriented Approach

The object-oriented paradigm has been touted as one of the most significant developments in the software industry in recent years, having the potential for significantly improving both the quality of software and productivity in developing it. Instead of emphasizing procedures and their impact upon data structures, with object-oriented techniques, an application domain area is viewed as objects, each of which has characteristic behavior. As with traditional analysis techniques, system developers must have an understanding of the problem to be solved. However, the emphasis is now on those entities which comprise the domain, both tangible and conceptual, and on their interactions. By shifting the focus to an application's objects, the developers can identify and represent system dependencies which may have been overlooked in a purely top-down manner. By capturing these objects as software components, their evolution and reuse in other development projects can be encouraged and rewarded. Revisions or changes to the system requirements affect the objects and their interactions; some changes can be accommodated more easily using the object-oriented approach. Enhancements can often be made without affecting the existing structure of the system. Instead, new objects, interactions, or behaviors can provide this increased functionality.

Even with all these advantages, there are some types of applications for which the object-oriented approach is not best suited. For applications which are highly algorithmic or procedural, or require lots of mathematical computation, the conventional approaches are still probably best suited.

The life cycle for object-oriented software development has the same basic organization as that of traditional structured analysis and design methodologies. System developers study the problem domain and define the system requirements; these requirements guide system design. The system is implemented according to the design specifications. System testing is performed throughout the development life cycle, but unlike the structured approach, development is very incremental and iterative. Another difference between the paradigms is the perspective of the developers throughout the life cycle, the roles and responsibilities involved in software creation, and the results obtained from each stage of software development.

Object-oriented software development has evolved unevenly. While there are many object-oriented programming languages, until recently comparatively little attention has been paid to the analysis and design of object-oriented software. Several methodologies have been developed to assist in software development and there are subtle differences among them. Some methodologies do not distinguish between analysis and design activities. In the remainder of this chapter we attempt to differentiate the various phases of object-oriented software development and discuss the most prominent approaches.

2.3. Object-Oriented Analysis

Object-oriented analysis (OOA) identifies and expresses the components of a problem domain in terms of objects, relationships among objects, and their behavior. Since objects consist of both data and the procedures which manipulate that data, object-oriented analysis must consider the structure of objects, the means by which objects interact, and the procedures which constitute an object's behavior. Several methodologies have been proposed for object-oriented analysis. Some identify the objects first and then define behavior (e.g., Shlaer and Mellor [SHLA88], Coad and Yourdon [COAD91a], and Rumbaugh et al. [RUMB91]). Others analyze behaviors first (e.g., Object Behavior Analysis [GIBS90]) and use that to identify the objects in a domain. All would stress the importance of understanding the problem domain before attempting to identify objects and their behaviors.

There are different approaches to identifying and defining objects. Some methodologies recommend that individual instances of a problem domain be specified and then abstracted as classes. Coad and Yourdon and Rumbaugh et al. distinguish between an object and a class, but Shlaer and Mellor define an object as an abstraction of a set of entities, i.e. a class. Each of these methodologies use objects to represent either tangible or conceptual entities. For example, Shlaer and Mellor [SHLA88] propose five types of objects. Tangible objects are perhaps the most easily recognizable (for example, letters, parcels). Organizations or individuals can also be represented as objects (e.g. the mailroom), as can events (a mail shipment). Interactions between objects may themselves result in the creation of new objects (legal contracts). Even object specifications can be viewed as objects (shipping requirements). While other types of objects can be represented, this sampling of objects demonstrates the possible variety of an object population. One guideline that Shlaer and Mellor use during object definition is that an object should have attributes; otherwise, it probably isn't an object.

Once an object has been identified, its attributes can be defined. Attributes are those characteristics used to describe an object and can represent either physical or conceptual characteristics. Attribute definitions can indicate whether the analyst has achieved a proper degree of abstraction. If several class definitions share attributes, these classes can be abstracted to form a new class which is a superclass. The previous classes can then inherit these attributes.

Aggregation defines objects to be comprised of other objects. Typically, one or more objects serve as components of a larger assembly. This type of relationship complements inheritance. Shlaer and Mellor do not address the concept of aggregation, thereby limiting the expressiveness of objects in their methodology, but both Coad and Yourdon and Rumbaugh et. al. accommodate aggregate objects.

After the objects and their attributes have been identified, the interactions between objects can be defined through the use of state diagrams. *State diagrams* are pictorial representations of the relationship between events taking place in the system (e.g., a help key being pressed) and the state of the system after the event has been processed (e.g., a help screen is displayed). (See Figure 2-1.) Coad and Yourdon, Shlaer and Mellor, and Rumbaugh et. al. all use some form of state diagrams to model the interaction among objects, but they use different terminologies. Coad and Yourdon represent these activities in a service layer. Rumbaugh et. al. places this activity in the dynamic modeling phase. In each case, the diagrams portray a system's functions in response to information received from external sources.

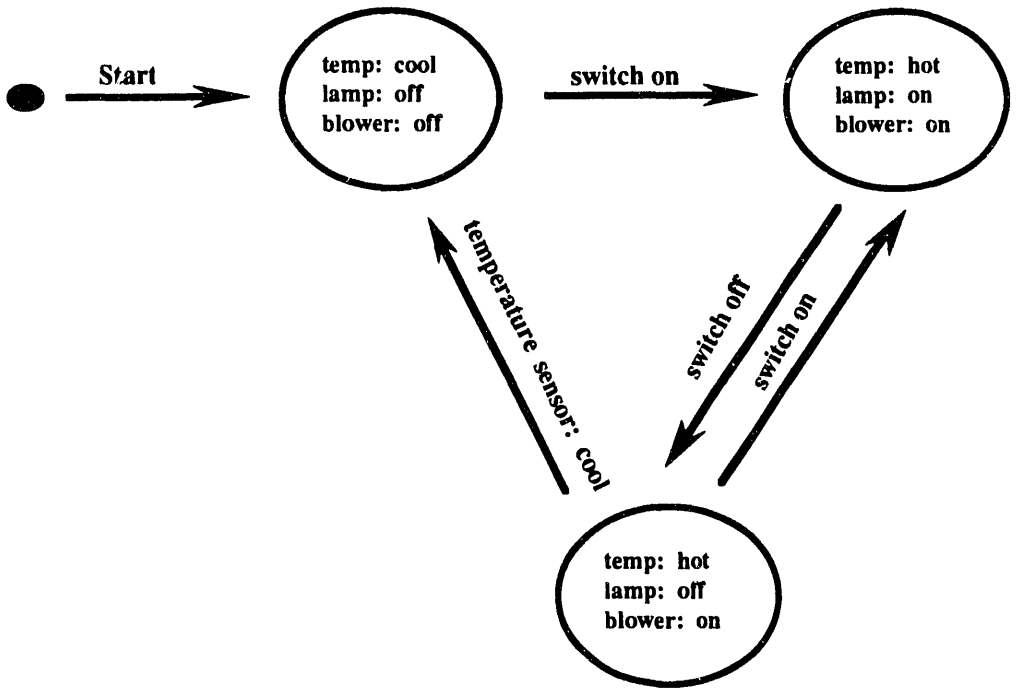


Figure 2-1. State transition diagram for overhead transparency projector.

Coad and Yourdon consider the Shlaer and Mellor approach to be more representative of semantic data modeling than object-oriented analysis. Their principal criticism is that these methods do not explicitly deal with object behavior. Shlaer/Mellor and Rumbaugh both employ state transition diagrams to "formalize behavior over time." Actions resulting from state transitions are represented as processes in data flow diagrams. Coad and Yourdon contend that this dependence on functional specification techniques ignores fundamental elements of the object-oriented paradigm.

There are some diagramming tools which support this process. For example, OOATool is offered by Object International of Austin, Texas. It provides drawing and validation support for Object-Oriented Analysis as defined by Coad and Yourdon.

The last portion of the analysis process that we will discuss is the specification of object behavior. Having defined a structure for the objects and identified the interactions among objects, the behaviors which are performed as a result of the interaction remain to be defined. Rumbaugh considers this the functional modeling process. In this activity, traditional top-down approaches can complement the object-oriented paradigm.

Most OOA methodologies concentrate upon identifying those entities which comprise a particular domain; having done this, they model the relationships among objects. Gibson [GIBS90] proposes Object Behavior Analysis (OBA) as a means of defining those behaviors to be exhibited by an application and deriving objects using this behavioral perspective. Objects are then grouped according to similar behaviors and properties. According to Gibson, the result is a requirements specification which "emphasizes the reusable aspects of a system, i.e., the behavioral protocols and the hierarchical groupings of objects according to such protocols." The highest level of

behavior defines the system's responsibilities; presumably, lower levels of behavior correspond to the responsibilities of collaborative objects. A variant of the CRC (Class-Responsibility-Collaborators) cards [CUNN86] is used in defining objects based on behavior. There is no explicit reference to inheritance and the methodology does not appear to have been used to develop actual applications. These and other object-oriented analysis methods are compared in a recent report from Hewlett Packard Laboratories [DECH91].

2.4. Object-Oriented Design

Object-oriented design (OOD) refines object definitions and specifies data management and human-computer interaction. Booch [BOOC91] and Coad and Yourdon [COAD91b] both discuss object-oriented design. Booch does not distinguish between analysis and design activities and views the development process from a higher level of abstraction than Coad and Yourdon. For application developers learning to design object-oriented software, Booch provides better insight into the motivations for the technology while Coad and Yourdon present a step-by-step approach.

As guidelines, Coad and Yourdon give the following steps for object-oriented design (OOD) using the results of object-oriented analyses: design the **problem domain** component, then the **human interaction** component, then the **task management** component, then the **data management** component.

The **problem domain** component uses the OOA results and improves upon them. Reuse of existing software components is encouraged and existing classes are specialized to obtain desired behaviors. The **human interaction** component includes the design of a detailed user interface, i.e., the actual displays and inputs needed for effective human-computer interaction. Windows, menus, text boxes, etc. are all specified in this stage. The **task management** component considers task definition, communication, and coordination. It considers whether the system is multi-user or multi-platform. The **data management** component includes access and management of persistent data. It isolates DBMS concerns, asking for example whether object and relational database systems are being used.

It is in the design phase of development that reuse of software components occurs. Reuse can be achieved by using existing requirements, designs, source code, or test suites for software. Class libraries, either customized or vendor-supplied, can substantially reduce programming efforts. It has been suggested that class libraries follow some convention to accommodate class reuse and to minimize or eliminate altogether inconsistent class definitions. One such scheme [MOOD91] organizes libraries into the following hierarchy: *Foundation classes* contain those low level entities upon which all other classes are built, e.g., string, stream. *Framework classes* provide those higher level entities which are employed by all applications, e.g., windows, buttons, menus, scroll bars. *Add-on classes* implement capabilities which may be needed for certain applications, e.g., databases, spreadsheets, matrices, graphics. Lastly, *application classes* are those classes developed specifically to meet the business needs of an organization. Figure 2-2 shows the different layers with sample classes for each.

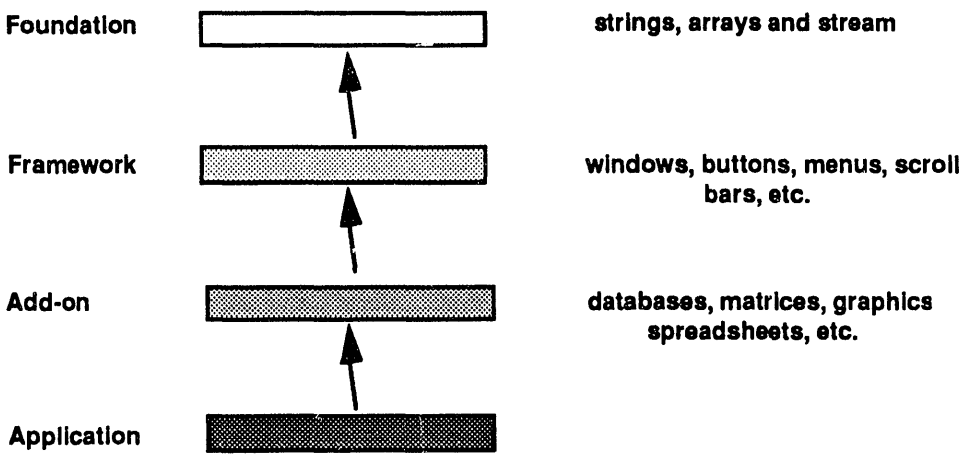


Figure 2-2. Hierachy of class libraries.

Another approach to object-oriented design is *responsibility-driven design*, which views an application as a collection of objects that collaborate to discharge their responsibilities, i.e., to perform the computations required by the system design [WIRF89]. The responsibilities of an object are those services that it provides for all objects that communicate with it. An object fulfills its responsibilities by performing some calculation or by collaborating with other objects. In this view, a *contract* is a set of related responsibilities defined by a class; this contract governs the interaction between collaborative agents. Behavior is formulated in terms of contracts, responsibilities, and messages. In this approach, design is partitioned into exploratory and analysis phases. During the exploratory phase, the classes required to model a domain are identified, the behavior of the system is specified, the responsibilities of respective classes are determined, and the collaboration among classes of objects is defined. Collaborations among objects are then identified. Once an initial model of the domain has been constructed, it must be revised to identify abstract classes, exploit reusability, and simplify inter-object communications. The resulting design should minimize the number of collaborations between classes of objects, limit the amount of delegation between them, and reduce the number of different contracts supported by each class of objects. Several object-oriented design approaches are compared in a recent report from Hewlett Packard Laboratories [ARNO91].

2.5. Object-Oriented Programming

The one common element among all object-oriented programming languages (OOP, OOPLs) is the notion of an object as the primary component of all programs. In contrast with the separation of procedures versus data in conventional programming, the notion of objects unifies data and process. *Encapsulation* protects objects, in that other objects may interact with it only through an interface. The internal implementation of an object is inaccessible. Most "pure" object-oriented languages enforce full encapsulation while extensions to existing languages rely upon encapsulation by convention. That is, programming style, instead of language constructs, dictates that an object's attributes will be accessed by those mechanisms defined specifically for that purpose. This form of information hiding via fixed interfaces promotes maintainability in that the

internal details of an object can change without affecting the manner in which other objects interact with it.

2.5.1. OOP Mechanisms

A *class* is a description of the structure and behavior of one or more similar objects; these individual objects are called *instances* of that class. Some object-oriented languages do not differentiate between classes and instances. Classes can be arranged in a hierarchy such that classes at a lower level of the hierarchy can share procedures and data from one or more classes at a higher level. See Figure 2-3 for an example set of three objects in a hierarchy. This sharing of data and procedures is called *inheritance*, and object-oriented programming languages support either single or multiple inheritance. Single inheritance allows a class to inherit data and procedures from only one parent class and its parents up the hierarchy; multiple inheritance permits a class to inherit data and procedures from several parent classes and all their ancestors. Shared data and procedures can be localized to avoid redundancy by placing them at a point in the hierarchy where they will be inherited by all classes that need them. A new class which is similar to an existing class can be added to the system by making it a subclass of the existing class, *specializing* it as needed by extending or slightly modifying its inherited data and procedures.

Objects have *methods*, or internal procedures, which define the behaviors they can perform. Most object-oriented languages initiate activity in a program by having one object send a *message* to another. A message typically consists of a selector which specifies the operation (method) to be performed and any appropriate arguments for passing needed information. A message of a given name may be sent to a variety of objects. Local message interpreters determine which actions to take, based on the class of the receiver. Thus a "Display" method would be implemented differently for a text object and a video object. Given these interpreters, uniform interfaces can be established between sets of objects by establishing standardized message formats. Such interfaces, called *protocols*, support data abstraction since a message sender need only concern itself with what the recipient is supposed to do, not how it is to go about doing it. The protocol must be general enough to permit modifications or enhancements at a later time (thereby promoting reusability).

Figure 2-3 shows how the information and behavior in objects can only be accessed via a message selecting the appropriate method to perform. In the example, the only access to objects A, B, and C is by using the four methods listed in objects B and C. In order to respond to the incoming "MoveTo" message to A, the MoveTo method code (pointed to by object C) is inherited and executed.

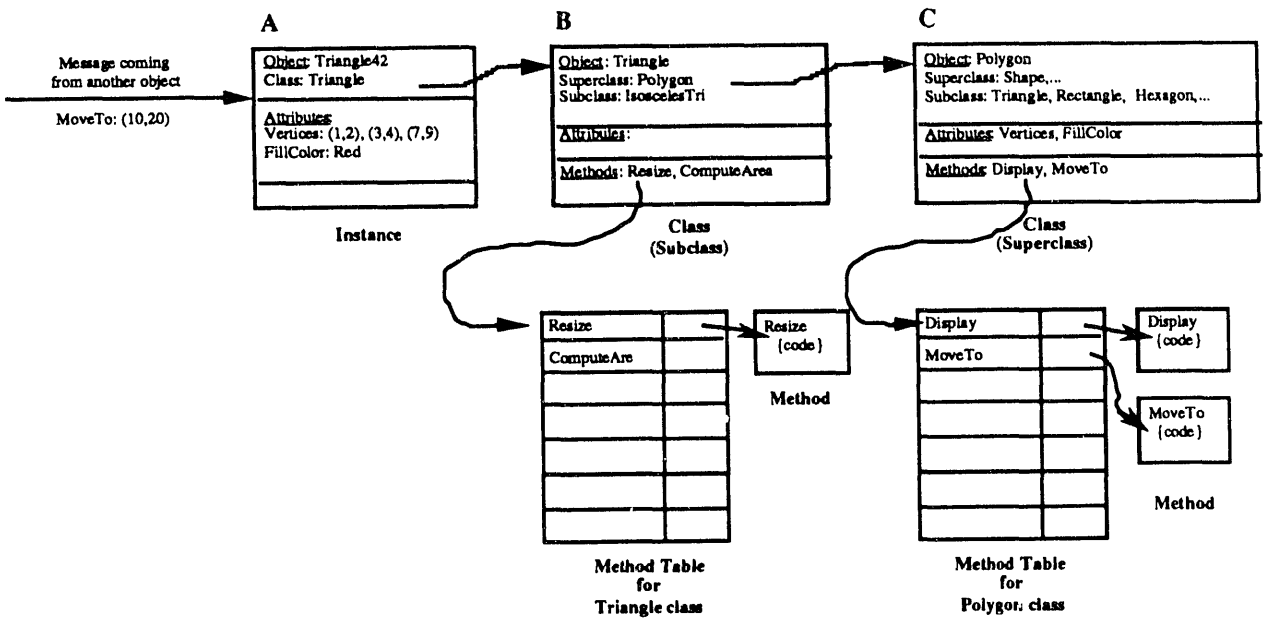


Figure 2-3. Object classes, methods, and message.

Message sending takes advantage of other features of object-oriented programming languages. *Polymorphism* is the capability of several classes of objects to respond differently to the same messages. Polymorphism depends upon dynamic or late binding. *Dynamic binding* determines at execution the correct function call reference for a particular object; conversely, static or early binding establishes this reference at compile time. While dynamic binding is not as fast as static binding, it gives the software developer greater flexibility. Dynamic binding and polymorphic functions promote software reusability by allowing the same function calls to be used by different objects with different structures and behavior.

2.5.2. OOP Benefits

Object-oriented programming enables software systems to deal with more *complex and heterogeneous data* than has been possible previously. This has made much more tractable the problems of manipulating sophisticated graphic interfaces, design and manufacturing information for parts production, and multimedia information of various sorts. *Maintainability* is enhanced by the localization of data and procedures, and enables program updates to be made efficiently since they need to be made in only one place. *Extensibility* is provided through customizing existing objects with more specialized attributes or behaviors to fit a specific application, and enables extensions to the program (e.g., new object definitions) which are slight variations of the existing system to be made with minimal effort and impact on previously written code. The working code remains unchanged, but the application has been extended or refined. These benefits allow us to isolate changes and minimize their impacts. This should also help to reduce resistance to making changes in software when they are needed.

Data *uniformity* is provided through the encapsulation of all data as object attributes, and procedure uniformity is provided through protocols. Both make programs cleaner and easier to understand. The *reliability and testing* of object systems are generally improved because the units can be tested

out completely when they are created, and that quality remains with the unit as long as it is encapsulated and used as expected. Having defined usage protocols for interfacing between objects yields better testing because there are fewer failure modes and types of interaction to validate. However, the powerful dynamic binding feature can also cause thorny testing and debugging problems, especially in interpreted systems such as Smalltalk. In a system of complex interactions among objects, it may be impossible to anticipate the sequence of messages which will be sent, and what the resulting states of the objects may be.

The recognition of software life cycle entities as *reusable components* leads to their treatment as *corporate assets*, and creation of a different mindset in thinking about software. These assets have great value to the Corporation, and this realization attaches significant importance to proper management (creation, storage, indexing, facilitation of retrieval and reuse) of these assets. The inventory of components becomes the organization's store of expertise, and commercial libraries (generic or specific to application areas) can be assessed and acquired where there is sufficient value added.

2.5.3. When is a Programming Language Object-Oriented?

Most languages purporting to be object-oriented allow the definition of classes and instances, inheritance of attributes or properties from parent to child, dynamic binding, method invocation, and some degree of data specialization. Using the definitions from a seminar run by the Association for Computing Machinery [HERK91], a language is *object-oriented* only if it:

- supports objects that are **abstractions** with an interface of named operations (**methods**) and a hidden local state (**encapsulation**)
- can associate one or more abstractions with any given object (**polymorphism**)
- allows abstractions to **inherit** attributes and behavior from other abstractions.

A language is *object-based* if it provides all the capabilities listed above except for inheritance. An example of this is Ada. An object-oriented language is "pure" if all constructs within the language are objects with associated classes. An object-oriented language is a "hybrid" language if some constructs are not objects.

Examples of pure object-oriented languages are Smalltalk and Eiffel. Two hybrid languages which supply object-oriented extensions to existing languages are C++ and CLOS (Common LISP Object System). Since hybrid languages do not enforce encapsulation, developers can implement much of their design using techniques which do not allow for extensibility or reusability. On the positive side, such hybrids permit a more gradual adoption of object-oriented methods within organizations.

2.6. Object-Oriented Database Management Systems

Objects created with an OOPL are transient; they disappear after the program terminates. However, objects could also be *persistent*, so that they could be stored and retrieved after the program restarts. However, the complex structures built during the execution of these programs cannot be directly stored to conventional database management systems, such as relational DBMS. One approach is to "flatten" the objects, breaking the complex data structures and procedures into tabular forms. Another is to create object-oriented database management systems (OODBMS).

These systems permit the objects to be manipulated as entities, supporting the encapsulation critical to the implementation of object-oriented programs.

The design of OODBMS has taken two differing paths: in one, object-oriented programming languages have been extended to support persistent objects and other database functions; in the other, existing database technology has been extended to permit the storage of complex data types, including procedures. OODBMS will be discussed in more detail in Chapter 5.

2.7. A Hybrid Approach

One of the advantages of the object-oriented paradigm is that an analyst can represent a problem domain in a natural form, as representatives of real-world entities and their behaviors. There is no need to transform the problem into a series of procedures. This representation produces a more understandable system for both customers and developers, since it is easier to see how the problem domain was mapped into software. Although object-oriented analysis and design techniques can be employed separately from implementation, much of this advantage is lost since the transformation to procedure-driven software components is merely delayed until implementation. Coad and Yourdon in their book are perhaps too optimistic about this:

> "If you receive a non-OOA requirements specification, rapidly develop (say over 1-4 weeks) an OOA model .... If you receive a non-OOD specification, yet you plan on implementing the design using an OOPL, rapidly develop (1-4 weeks) an OOA model, and expand into an OOD model. Finally, apply detailed OOD ...." [COAD91a, pp. 181-182]

This suggests, perhaps unrealistically, that functional specifications can be quickly translated into object specifications. Other authors are much more pessimistic about the ease of making the transition from structured to object-oriented paradigms.

The use of an object-oriented programming language which supports the notions of abstraction, inheritance, and polymorphism greatly simplifies the translation of the object-oriented design specifications into source code. Both Coad and Yourdon [COAD91a] and Rumbaugh et al. [RUMB91] indicate that object-oriented analyses and designs can be implemented in procedural programming languages, but that object-oriented programs are easier to write, maintain, and extend. Rumbaugh describes an approach for translating object classes to relational tables. The need for persistent object storage can be more easily satisfied by the use of an object-oriented database management system, if use of one is feasible in the given computing environment.

Chapter 3:

OOT and Applications

3.1. Introduction

We will discuss several major application areas that may benefit from object-oriented technologies: design/manufacturing systems, modeling and simulation, management information systems, and multimedia information systems. As object technology improves, increases in time and space efficiency of the software tools will make it realistic to also deliver some of this work in actual real-time environments. However, the scope of this paper excludes the problems of real-time systems.

Over the last two decades, computer automation has been promoted as the means of providing the increased productivity needed to improve a company's position in the competitive market. Some applications such as accounting, scheduling and material planning are well understood manual processes and have been successfully automated. However, the resulting productivity gains from automation have not materialized to the levels promised. This is primarily due to the complex nature of the products and processes used. Designing, manufacturing and marketing competitive, quality products requires precise coordination and integration of all of a company's resources.

While there are many successful automated CAD/CAM/CAE applications, the net effect on the enterprise has been to create what is commonly referred to as "islands of automation". Each application is built around its view of the product and the processes it automates. These applications have been very successful at automating their specific view of the product but in the process have isolated the information they use and create from the other applications in the product life cycle. The major problem with software from the commercial-off-the-shelf (COTS) vendors is that the COTS software encapsulates the data it creates and uses in a relatively impenetrable wrapper. Most CAD/CAM/CAE software has been written from the perspective of a stand-alone application and not as a supporting piece in the full life cycle of a product. Vendors have developed their own private application-driven databases structured for efficiency, but have sometimes not even been able to integrate their own application software effectively.

The management of bill-of-material (BOM) information throughout a product's life cycle is an example of an application for which the object-oriented paradigm appears ideally suited. Each life cycle stage has its own viewpoint. For example, in the conceptual stage, a BOM may be a list of requirements; in the design stage, it could be the list of components that are assembled to make up the product or it could be the a list of the materials needed to make the entire part; in the manufacturing stage, a BOM might be the list of materials needed to accomplish an in-process manufacturing activity and so on throughout each stage. The object-oriented approach allows one to identify object and application view subclasses for each of the specific types of BOM. Each can be structured and behave according to its role in the life cycle. Encapsulation, multiple inheritance, polymorphism, extensibility and persistence are all useful characteristics that could effectively be utilized to support this multi-faceted application.

Engineering data are in general more complex in structure than business data. Traditional software and computer technologies have not been able to deal adequately with complex product life cycles, design and manufacturing processes, or with sophisticated human interactions using hypertext and

graphical user interfaces. This difficulty may be partly due to the lack of real-world physical models to understand and emulate. Using the methodology and tools being developed within the object-oriented paradigm, the long-promised increased productivity obtained through automation may now be achievable.

Another area of advanced information processing is that of simulating and modeling complex systems. Object technology allows the efficient extension of typical simulation package capabilities to model dynamic systems of various sorts. Several such systems will be discussed later in the chapter. Another application area where objects provide a very natural representation mechanism is in modeling and managing communication networks. A network management tool based on object technology is being evaluated for use in Oak Ridge.

Traditional database management systems (DBMS) have proven to be effective tools for organizing and maintaining large volumes of data. DBMS technology allows multiple users to share the same data in a timely and sensible fashion. In many enterprises, DBMS are the focus of new and varied applications. Most of the successful DBMS applications have been oriented towards processing business data. The primary reason for this is that the traditional DBMS systems are capable of operating in a relatively efficient manner on massive volumes of data where the data structures are relatively predictable and easily understood. Usually there are few data traversals required for each transaction and the number of different transaction types is small.

Industry trends are driving the need for information management systems capable of representing and modeling the behavior of large, complex, *multimedia* data types. Many types of automated information have become available; we now have sound, still image, video, and graphic and CAD data as well as traditional text and numeric information. The required information management cannot adequately be provided with existing relational data base technology. Object-oriented technology offers the promise of removing some of the existing technological barriers. OOT supports the ability to navigate through heterogenous types of data as well as the ability to process queries on sets of homogeneous data. With this technology it is possible to design systems that have mechanisms for change management, managing group work and sharing and permanently storing objects for use in multiple applications. The defining characteristics of OOT (encapsulation, inheritance, polymorphism, extensibility, objects, persistence, class libraries, etc.) are the types of concepts that are needed to establish and implement these kinds of complex information management systems.

This chapter includes both generic discussions of several types of applications and descriptions of specific projects. Some of these projects are described in more detail in the Appendices. Across the DOE complex there has been a substantial amount of work using mostly C++ and some Smalltalk, and a few projects have used commercial object-oriented database systems. Although the technology is still young, we are already developing significant experience and expertise.

3.2. Design and Manufacturing Systems

Design and manufacturing are application areas that generate data processing requirements which exceed the capabilities of existing DBMS. There has been a tremendous amount of time and money expended throughout the public and private industrial sector to automate and integrate the activities and processes occurring throughout the life cycle of a manufactured product. A major business objective is to significantly reduce the time it takes to conceive, design, manufacture and market a product while reducing its cost and improving its quality. In order to accomplish this, the general

consensus is that the life cycle activities of a product will have to be automated and integrated in a data-driven environment centered around an intelligent product information database. There are several emerging design and manufacturing methodologies which will help to achieve these goals:

- Total Quality Management (TQM)
- Computer Integrated Manufacturing (CIM)
- Design for Manufacturability (DFM)
- Design for Quality (DFQ)
- Concurrent Engineering.

For success, these methodologies require that an integrated data-driven environment be established within the enterprise. Many attempts have been made to design and implement such a workable database with little success because of a number of problems. One of the technical barriers, which is now dissipating with emerging standards such as STEP (Standard for Exchange of Product data), has been the absence of a unified standard for representing the complex relationships, constraints and rules which exist among product data. However, it has also been established that existing relational DBMS technology is not sufficient for handling massive databases where the structures are complex, the database transactions are long and complicated, and the number of different transactions is extensive. Evidence of this is that there are no commercial CAD systems whose working database is truly relational.

It appears that there will be commercial object-oriented DBMS capable of creating cost-effective database environments for:

1. describing product life cycle information including design intent and the manufacturing processes.
2. reusing, exchanging and managing engineering, manufacturing and support information.
3. supporting the common sharing of complex design and manufacturing data and the cooperation, collaboration, and coordination of these groups.
4. integrating with existing relational databases, CAE/CAD/CAM and image applications, and languages.
5. supporting standards such as CFI, CALS and PDES/STEP, and Open Systems.

It is generally conceded that the compromises necessary to use current relational DBMS technology yield systems which are too slow and unwieldy to support these types of complex databases in a concurrent environment, so great hope is placed on the emerging OODBMS products.

3.2.1. Some DOE Manufacturing Projects

At Allied-Signal in Kansas City, there is a whole family of advanced manufacturing projects under development which will communicate their data models via PDES representations and will share information using an object-oriented database system. The XCUT system (see Appendix G) will generate process plans which call out tasks for numerical control analysts and inspection planners. The advanced numerical control planning system ANC will scan the process plan for NC requirements, generate tool paths, and add them to the database. The inspection planning system IPPEX (see Appendix H) will scan the process plan for inspection requests and add inspection tasks to the database. These systems rely on the Advanced Manufacturing Development System AMDS (see Appendix I) to provide product design data translation from the design agency's model representation to Kansas City-specific representations .

Prototype systems are also being developed at Martin Marietta Energy Systems in Oak Ridge (described in Appendix J) and at other DOE sites. Although the use of object technology is quite promising, actual use of this approach for developing production manufacturing systems is currently limited to one system now being implemented (Appendix R). Widespread use of object technology is not likely for several years.

A project under way at Sandia National Laboratory extends the ideas behind design and manufacturing to the area of *planning*, specifically how to streamline a large cooperative planning process involving a number of government enterprises. The hypothesis is that using modern extended relational databases or object-oriented databases would not only allow concurrent planning operations but would also yield a number of other benefits (see Appendix K).

In Los Alamos there are a couple of projects which bridge the areas of process control and modeling. For the last couple of years, some earlier artificial intelligence work has been extended to develop a high-level modeling and process control system (see Appendix L). In a separate project, the KOALAS architecture for intelligent control systems combines conventional control theory with automated reasoning techniques (see Appendix M). This architecture is currently being applied to automated multi-sensor integration in tactical naval aircraft.

3.3. Modeling and Simulation

As any large organization strives to fully understand and improve its complex processes, and to make plans for future operations, modeling and simulation are widely used and important techniques. Object technology allows extending traditional approaches to modeling complex and dynamic systems.

One area of modeling which is of strong interest in the DOE Complex is in **nuclear reactor operations**. In Oak Ridge, an object approach was used to create a modeling tool called GOOSE (Generalized Object-Oriented Simulation Environment) for developing dynamic models of systems such as nuclear reactors (see Appendix N).

At Savannah River there is an ambitious plan for giving nuclear reactor scientists and engineers a scientific computing environment which will maximize their ability to manage their scientific data and use simulation to understand and solve the scientific and engineering problems associated with nuclear reactors. Object technology is being used to create a set of graphics tools (portable across systems) to enhance application development and human interfaces to the systems. See Appendix O for a description of this work.

There are a number of other modeling efforts across the DOE complex in the areas of **weapons production processes and waste management.** The Technology Assessment Selection Panel (TASP) model was started several years ago at Los Alamos using artificial intelligence tools and techniques (see Appendix P). The model now has involvement from several DOE sites (including a large effort at the Y-12 Plant in Oak Ridge) and is moving to an object-oriented set of tools. The purpose of the TASP model is to study DOE weapons complex production facilities and plan for the proposed Complex 21 reconfiguration. In a similar vein, the Rocky Flats Plant Simulator (RFPS) has been developed at the Rocky Flats, Colorado facility to help with planning and scheduling their people, equipment and processes (see Appendix Q). The early emphasis of the project was on manufacturing, aqueous recovery and pyrochemical processing buildings, but with recent mission changes, modeling efforts have been redirected to include stabilization and

elimination of liquids and other wastes. The RFPS is needed to help answer a variety of "what if" questions with the new mission of the Plant.

Again at Savannah River, object-oriented techniques and tools were used to model the Defense Waste Processing Facility (DWPF).[1] The DWPF will be used to process radioactive wastes into a glass substance which can then be disposed of safely. The system is composed of three programs. The Glass Compositions Program gives a visual display of how to mix waste components into glass components, under a number of constraints. The Batch DWPF Simulation Program provides a graphical, real-time simulation of a batch chemical process operation, keeping track of levels, transfer of material between vessels, and all the process steps (heating, cooling, boiling, condensing, etc.). The Product Composition Control System (PCCS) does statistical process control, monitoring all sample analyses and maintaining the status and sample assay of each vessel to know what is happening at any given time.

3.4. Management Information Systems

Management Information Systems (MIS) applications provide management, supervisory, and operating personnel with the business information required to effectively manage ongoing activities of the enterprise. Examples include human resource systems, activity scheduling systems, manufacturing support systems, and production measurement systems. A primary motivation for MIS application developers to consider object-oriented technologies is rooted in their pursuit of methods to maximize the correspondence between MIS systems and the business functions they are intended to support. Unlike engineering and manufacturing systems, their motivation is not linked with inadequate data management support from existing languages and database management systems. Another strong motivation for using OOT is the software engineering benefits which can be attained. A library of tested, high quality objects can be created and reused for modeling the information structure of the organization. This approach also reduces the high cost of maintenance and the system enhancements which are inevitably needed.

An interesting project developed in Oak Ridge used objects to address the needs of a "paperless office." This system, described in Appendix S, developed a secure electronic signature capability and incorporated that type of verification into an electronic document handling system.

The "First Priority" project at Savannah River used an object-oriented approach to build a generic prioritizing scheme (see Appendix T). The system builds up a tree-like structure which represents principal goals, subgoals and concrete measures which are then used to compare issues. The system has been used for a variety of prioritizing tasks, including responses to Tiger Team findings, request for personal computer systems, and selection of technical award winners.

3.4.1. The Problem With Traditional Approaches to MIS

In early approaches to the development of enterprise MIS applications, software developers discussed the requirements of the business with appropriate business representatives and translated

---

[1] A paper on this system "Control of DWPF Melter Feed Composition", by R. E. Edwards, K. G. Brown, and R. L. Postles of the Westinghouse Savannah River Company, was presented at the Nuclear Division of the American Ceramics Society meeting in Dallas, TX April 22-26, 1990. Contact Richard Edwards (803: 725-6456) for more information.

the requirements they heard into an application system design. Business users, however, found it difficult to express their expectations of a computer system with precision, and in terms that could be effectively communicated to software developers. The result was many application systems that were not responsive to the business needs. Formal specifications were needed - expressive specifications that both the business user and software developer were comfortable with. Elaborate methodologies were developed to manage the creation of these specifications. These methodologies were broad in scope, long on detail, and intractable for any but the smallest of MIS software projects.

Today's commercially successful CASE products address the specification problem by lending software support to many of those same methodologies. The most common methodologies are based on a modeling paradigm that treats the definition of data and of function as separate tasks. Data models are distinct from process models, and numerous methods, techniques, and tools are used to validate each model in the context of the other. The treatment of data and process as different entities represents the major disconnect between existing CASE products and the object-oriented approach. The effect of this disconnect is the perpetuation of the original problem -- miscommunication between business users and software developers.

The heart of the problem is the process modeling activity. There is no single, correct hierarchical model of business processes. At the top of the process hierarchy the decomposition is essentially arbitrary. The methodologies indicate that processes (often referred to as functions at the upper levels of the hierarchy) should be defined without consideration of existing enterprise organizational structure. Often this level of independence can be accomplished at the high level, but usually it breaks down as processes are successively decomposed. The forces that effect this breakdown are fundamental. Process decomposition independent of organizational structure is useful in defining and understanding the business, but application software systems are defined, developed, used, and most importantly funded, by **organizational** units. They require an operational design that reflects their organization's responsibilities and preferences. Organizations, however, are among the most dynamic entities of a business enterprise. Building their structure (even indirectly) into the model creates significant instability in both the business model and the software systems implemented to support it.

Information engineering methodology represents a transitional approach, in that the interrelation of business processes and data is acknowledged. Business processes may be defined by the manner in which they change the state of information over time. However, it is still difficult to avoid the injection of organizational units into the system definition. The defined relationship between data and process still falls short of that represented in object-oriented approaches.

3.4.2. Benefits Of An Object-Oriented Approach

The object-oriented paradigm represents a major shift in thinking for business users and MIS software developers. Object classes represent the fundamental entities to be modeled, and all processes are modeled in the context of an object class. This technique has many benefits:

- focusing on data first allows the model to be built on the most stable element of the business environment

- defining process or functions strictly within the context of a data entity prohibits organizational structure considerations from becoming part of the model at any level

- by postponing the operational definition of an application system to the last possible moment, developers are able to maximize the reuse of software components, and minimize investment in software that provides an organizational view of their update and query requirements.

3.4.3. Data Requirements For MIS systems

MIS systems require complex user-defined data types and functions, plus management of data aggregates and their associations. Current object-oriented technology can successfully meet these needs. But in the business environment there are often also requirements for multi-user concurrency management with cluster locking and for performance-oriented management of large volumes of structured data. These capabilities are generally not present in current OOT systems.

In recent years, the MIS area has moved away from the tendency to solve individual problems with independent systems, and is reaping the benefits of the data modeling approach using information engineering and CASE tools. Starting from the top, the enterprise as a whole is modeled, then each business area is analyzed in turn to yield business systems, and applications are developed. Object technology can be added to the computational architecture to help shape the systems built within the enterprise business model. But a widespread shift to the object paradigm to achieve the benefits of OOT awaits strong commercial support of object-oriented techniques by both CASE vendors and conventional database management systems vendors. We see that support coming, for example in Texas Instruments' statements about future releases of its IEF integrated CASE product. However, until that support is mature, there is little likelihood that most MIS developers will embrace the new object techniques.

3.5. Technologies which Benefit from Object-Oriented Technology

3.5.1. Multimedia Information Systems

Earlier in this chapter we mentioned the many types of automated data that are available for use today: not only text, but graphics, still images, video and sound. If a computer is used to integrate such multiple forms of communications media, then *interactive multimedia* systems can be built. These systems act as nonsequential documents, or *hypermedia*, and allow the user to interact with and control the flow of information. The user of the system can follow links in the web of available information, browsing freely or choosing specific information needed to solve some problem. One such system has been built in Oak Ridge to support technology transfer by representing the diverse capabilities of Energy Systems development and manufacturing capabilities and facilities. The system gives flexible access to information at all levels of detail and presentation in several formats (see Appendix U). Multimedia tools are quite valuable for illustrating concepts that are hard to describe in text, or events that go by too quickly or slowly for normal observation. They can bring up documents or regulations that apply to some event, while automating analysis and record keeping about the events. The systems can deliver sounds or visual images that enhance the information needed for some task.

Almost every application domain area can benefit from use of multimedia information; people keep creating new applications for it. Multimedia is especially appropriate for procedural support/reference and for training systems. Several such systems have been constructed at Energy Systems. One early system was KATIE, built using Smalltalk, which used an expert systems

approach to guide users through the maintenance and diagnosis of complex manufacturing equipment. KATIE uses video displays of procedures being performed on the equipment, along with active text presentation of required procedures. A later version of KATIE (still object-oriented but not requiring expert systems techniques) is being used to train security force personnel. See Appendix V for more information on this system. Other training systems have been built using object-based products which make it easy to combine hypertext and visual information into a training presentation. See Appendix W for a description of a system which was built using Owl's GUIDE product.

3.5.2. Graphical User Interfaces

Perhaps the strongest driver for the growth of interest in OOT has been the development of graphical user interfaces (GUIs) on personal computers, notably those of the Macintosh, Windows 3.0, Presentation Manager (OS/2), and X-Windows (UNIX). According to Bill Gates, chairman of Microsoft, the goal of working with a computer is "information at your fingertips" -- the ability to find, compile, and summarize information from many electronic sources without even having to know where it comes from [VERI91]. A graphical interface gives a user much greater power over the interaction with a computer system and requires almost no training to use at a competent level. The desired type of interaction now has information presented in windows, with the user manipulating choices represented as icons by selecting them from a menu with a mouse or other pointer. This type of interface is called a WIMP interface (for Window, Icon, Menu, Pointer).

Provision of such interfaces is a real computing challenge, especially if the ability to scroll through a large region is included, and users are allowed to change the size of and move their windows. Several tools are available (such as Hypercard, Supercard, HP New Wave, and public domain Interviews from Stanford) which ease the development of WIMP interfaces. Use of an object-oriented approach was necessary to build these interface managers. Coad and Yourdon motivate the use of object-oriented approaches with the following statement:

> "software managers at IBM commented ... that all of their attempts to use standard structured techniques to build applications in the OS/2 Presentation Manager environment [were] dismal failures; the only successes ... were applications developed with object-oriented techniques." [COAD91a, p. 191]

All types of applications can benefit from WIMP interfaces. Even though the prognosis was somewhat pessimistic about near-term use of OOT in management information systems, use of objects to provide an interface to business information is already feasible and quite appealing. There is a special class of management information system called Executive Information Systems (EIS) which provides highly interactive management-level information about an organization. These systems benefit greatly from the use of object technology, largely because of the type of user interface they need.

3.6. Applications Which Are Not (Yet) Appropriate for OOT

There are some types of applications for which OOT is not yet mature enough to solve the problems, and others for which existing solutions seem better than what OOT could offer.

The business-oriented MIS applications described in the previous section are in the first category. The need for high performance over great transaction volume exceeds current OOT capabilities. Similarly, current object tools and object-oriented database management systems do not yet provide shared, secure multilevel access to classified or sensitive information. When these capabilities and performance levels are reached in the maturing products, then use of OOT for applications which need those features can be reconsidered.

In the other category, there are some types of applications which need algorithmic solutions which are already implemented. When this is the case and there are no problems with existing solutions, there is no reason to change the legacy applications to object technology. Indeed, if an existing system is to be integrated with a larger object system, one way to handle the interface is to use "wrappers" which treat the existing system as a "black box" object and use its standard calling sequence to invoke the system's behavior. In this way, there is no need to reprogram the system.

Chapter 4:

Object-Oriented Application Development

4.1. Need for Fundamental Reorientation

A basic problem with the structured approach to software development, according to object technology leaders such as author Brad Cox and consultant Jon Hopkins, is that it is **process-centered**, so that the programmer-machine interaction and the processes we use to create software are paramount. With software development in crisis, unable to keep up with hardware and telecommunications advances and cost reductions, we must abandon our tradition of hand-crafting software from raw materials. Instead, we should take a **product-centered** view of software, where the producer-consumer relationship and the software products needed by the consumer are more important. Cox argues that after the "software industrial revolution", programmers will choose standard, reusable components from software catalogs (class libraries) and combine or specialize them as necessary to create new applications.

We could draw an analogy from our experience in engineering electronics. Twenty to thirty years ago, our engineers designed any circuits needed for the devices we built. But as technology advanced, a variety of prefabricated integrated circuits with standard interfaces became commercially available. This enabled quicker and less error-prone assembly of complex devices, using off-the-shelf components.

Now a similar phenomenon is beginning to take place in the software industry. Once the software components market is established, consumers themselves may be able to assemble many needed applications from robust off-the-shelf components marketed by several layers of commercial software producers.

Although Brad Cox sees much hope with object-oriented technology to carry out the software industrial revolution, he warns that implementation tools to create interchangeable parts cannot succeed alone; there must also be **specification tools** to determine whether components are combined and behaving according to specification and within tolerance. In other words, you need not only a saw to cut a piece of wood, but also a ruler with which to measure before and after cutting a piece. The primitives of this needed specification language would be test procedures which would collect operational measures to determine software product quality. That is, the specification tool would monitor the consumer's interface with the product, instead of the more traditional focus we have had on the software producer's interface (using lines of code or complexity measures, for example). [COX90]

4.2. State of CASE Tools

The state of Computer Aided Software Engineering (CASE) tools today is that they are productivity enhancing tools, increasingly covering all life cycle phases, but mostly for developers building Management Information Systems applications, using structured methodology, and generating COBOL code. For other types of development, the state of the art is less helpful. The ultimate power of CASE tools is that they allow the developer to create the application at an abstract level by diagramming it, and then the tool can generate the code needed. If program modifications are needed, the developer is supposed to generate them through the high level representation of the CASE tool, not by revising the code. In practice for many technical organizations, the COBOL

generation is often not desired, so the full benefit of CASE tools is usually not achieved. However, this situation will change as more and more CASE tools generate C code.

The abstract, high level representations of an organization's applications and data stores must be stored permanently in *repositories* to allow modification or extension to other applications. There are some tool-specific repositories, but CASE tool users naturally would like an open, common repository that could communicate with any vendor's CASE tool. A number of vendors are working on repositories (e.g. IBM, DEC, Texas Instruments, KnowledgeWare, Oracle, Computer Associates). IBM's emerging AD Cycle/Repository has received much attention in the trade press, but it is not clear when it will be successful. The nature of the application diagramming methodology and the representational capability of the repository determine the types of applications that can be generated; that's why most current tools generate structured COBOL. A CASE tool for object-oriented applications must have a repository that can store class and instance objects and their methods, and that sort of capability doesn't easily coexist with the structured tools.

The key difference between the structured and object-oriented systems is that conventional CASE tools generally use a procedural environment to represent the applications, whereas object-oriented (and knowledge-based) application development tools tend to use more declarative and business domain-focused information. "The more generic, declarative, and business-focused the representation, the more powerful and flexible the tool", says industry watcher Paul Harmon [HARM91c]. But although the declarative representations are more powerful and will increasingly be the preferred tools during the 1990's, a limiting problem is the lack of an accepted methodology for object-oriented development. There are several such methodologies being developed, but their ability is still pretty limited. Although AD Cycle/Repository does have some storage capability for object-oriented components, IBM has postponed its scheduled announcement of AD/Cycle support for OOT. Instead of announcing support for AD/Cycle with at least one object language (probably Smalltalk and perhaps also C++), IBM decided to delay the announcement and address the larger issue of OOT and how it will affect the whole software life cycle. On a less optimistic note, Harmon predicts that the AD Cycle/Repository will not be viable even for conventional CASE use before 1995.

Many CASE tools already use object techniques behind the scenes to help provide the structured application development capability, usually in the graphical interface. Very few tools allow developers to create object-oriented systems and generate C or C++ code. (See Table 4-1.) Of the conventional CASE vendors, Harmon wrote that Texas Instruments' IEF tool seems best positioned to develop an Intelligent CASE tool, adding objects as well as a knowledge-based systems capability. Table 4-1 shows the comparative capabilities of six popular CASE tools [HARM91c].

Table 4-1a. Popular CASE Tool Vendors[1]

Product Company	Application Development Workbench (ADW) KnowledgeWare, Inc.	Excelerator Series Intersolv Inc.	Information Engineering Facility (IEF) Texas Instruments
Basic Functionality of product	Enterprise modeling Analysis and design of applications Interpreted RAD/Prototyping environment Code generation (database generation) Application and database design utilities Reverse engineering	Enterprise modeling Analysis and design of applications Code generation (interface code generators) Reverse engineering	Enterprise modeling Analysis and design of applications Interpreted RAD/testing environment Code generation (database generation) Test and maintenance utilities Reverse engineering Prototyping
Development OS/ hardware	IBM PC, DOS; PS/2 OS/2 Standard or EE 1.2 or >, MVS (min. 8 MB memory recommended). Presentation Manager interface.	PC AT, PS/2, DOS.	IBM AT, MS DOS 3.1 Or >; PS/2, OS/2EE 1.2 or >.
Delivery OS/ hardware	IBM PC and PS/2, OS/2, MVS.	PC AT, PS/2, DOS	OS/2 wk/sta., OS/2EE 1.2 and DBM; DEC VMS; Unix Sys. V (TI 1500 and Fujitsu). IBM MF, MVS/XA, MVS/ESA with DB2.
Price	Starts at $20,000 for 4 workstations; individual workstations start $10,125.	$9,800	Workstation toolsets; $9,400 - $23,000 Mainframe: $100,000 - $340,000.
Development methodology supported	Methodology independent; however, also supports: Martin, DeMarco/Gane - Sarson, and Yourdon. Customizable for supporting in-house methodologies.	Methodology independent; however, also supports; Yourdon, Ward - Mellor, Gane - Sarson, 4Front, etc. Customizable for structured development.	Information Engineering Methodology
Database/Repository Supported	IBM AD Cycle/Repository compliant. (IBM AD Cycle development partner). Integration with DB2, Focus, Oracle, Progress, RBASE, Revelation, etc.	XL/Repository (proprietary); IBM AD Cycle/Repository compliant.	IEF Central Encyclopedia (proprietary repository). Integration with DB2, OS/2 DBM, Rdb, and Oracle.
Language tool was initially written in	C, Prolog	C	C, PL/1, ALC
Underlying techniques used in product	Object-oriented design and analysis Information Engineering Structured analysis Structured design	Object-oriented technology	Object-oriented technology Rules and inferencing Pattern-matching technology
For more information contact	KnowledgeWare, Inc. 3340 Peachtree Road, N.E., Suite 1100 Atlanta, GA 30326 Phone: (404) 231-8575	Intersolv Inc. 3200 Tower Oaks Blvd. Rockville, MD 20852 Phone: (301) 230-3000	Texas Instruments Advanced Info. Management Division 6550 Chase Oaks Blvd. MS 8474 Plano, TX 75023 Phone: (600) 527-3500

[1]Source: Paul Harmon, ed., *Intelligent Software Strategies,* Vol. 7, no. 4, April 1991. Used with permission by Associate Editor Curtis Hall.

Table 4-1b. Popular CASE Tool Vendors[2]

Product Company	Maestro Softlab, Inc.	Teamwork Cadre Technologies, Inc.	Telon, Telon PWS Pansophic Systems, Inc.
Basic functionality of product	Enterprise modeling Analysis and design of applications Interpreted RAD/testing environment Code generation Test and maintenance utilities Reverse engineering	Enterprise modeling Analysis and design of applications Interpreted RAD/testing environment Code generation Test and maintenance utilities Reverse engineering	Enterprise modeling Design of applications Interpreted RAD/testing environment Code generation (database generation) Test and maintenance utilities
Development OS/ hardware	IBM AT, IBM RS/6000 or 9000. Client server architecture using MS DOS (client) and Unix (server).	Apollo, Sun, DEC, IBM, HP, DG, and OS/2; Unix, Ultrix, AIX, VMS, HP-UX, Domain OS/2.	IBM Mainframes, MVS. IBM AT or 386 (min. 4M B ext. memory), MS DOS 3.0 or >; OS/2 1.2 EE or >.
Delivery OS/ hardware	Any IBM, DEC, HP, Honeywell Bull, Unisys; any Unix	Apollo, Sun, DEC, IBM, HP, DG, and OS/2.	OS/400, MVS: CICS, IMS/DC.
Price	Approximately $15,000 per workstation.	Pricing starts at $6,500; 50 workstations approximately $650,000.	Telon: $150,000 - 450,000 Telon PWS: $2,250 - 12,500.
Development methodology supported	"Any methodology can be supported." SA/SD, SSADM, Merise, Selec implementation available.	Supports: Yourdon - DeMarco, Yourdon - Constantine, Booing - Hatley, Buhr OOD, Ward - Mellor, Martin - Schalaer OOD.	Methodology independent; RAD
Database/Repository Supported	Developed for DBMS. Uses Softlab OMS (Online, Multi-user OODBMS) as repository; others supported, including IBM AD Cycle via interface.	Proprietary database and repository	Proprietary database and repository; IBM AD Cycle/Repository complaint.
Language tool was initially written in	C	C, C++	Assembler, COBOL
Underlying techniques used in product	Object-oriented technology Rules and inferencing Pattern-matching technology Client-server architecture Open systems	Object-oriented technology Rules and inferencing	Object-oriented technology Rules and inferencing
For more information contact	Softlab, Inc. 188 The Embarcadero San Francisco, CA 94105 Phone: (415) 957-9175 World Headquarters, Germany	Cadre Technologies, Inc. 222 Richmond Street Providence, RI 02903 Phone: (401) 351-5950	Pansophic Systems, Inc. 2400 Cabot Drive Lisle, IL 60532 Phone: (708) 505-6000

[2]Source: Paul Harmon, ed., *Intelligent Software Strategies*, Vol. 7, no. 4, April 1991. Used with permission by Associate Editor Curtis Hall.

A number of organizations have adopted the Information Engineering (IE) methodology as a means for producing high quality software. Without the use of automated CASE tools to assist the software teams, carrying out the associated IE processes manually would be so expensive in time and resources that it just would not be done on many projects. The CASE tools for doing information engineering are fairly prescriptive in taking a project through the life cycle; there is not much flexibility for using different techniques. So, although some of the widely-used CASE tools are already using object-oriented techniques to develop their software, there is as yet no facility to do object-oriented development of applications. However, most of the CASE vendors are seriously studying OOT and making plans for adding object capabilities. CASE industry leaders such as James Martin seem to believe that OOT is definitely in the future of the conventional CASE products, and that the analysis and design phases of Information Engineering will change to accommodate objects. Data models will evolve to be more object-oriented, perhaps even allowing some mechanism to store methods along with objects, while the procedural components will endure as a parallel mechanism for many years. At the 1992 CASE World conference, Texas Instruments announced plans to use its proposed Information Engineering with Objects (IE/O) methodology in Version 6 of IEF, scheduled for 1993 release.

4.3. Object-Oriented Programming Languages and Environments

Object-oriented programming languages permit software developers to implement object-oriented designs using the concepts of data abstraction, inheritance, and polymorphism (see Appendix B for a description of the most popular languages). Although several of these languages have been available for years, there have been few attempts at standardization. Class definitions and method invocation may vary greatly from one language to another; consequently, programmers familiar with one object language may have difficulty with another.

There is much debate in the language community at this time about the relative merits of Smalltalk versus C++. Many developers argue that Smalltalk is easier to use and produces better object-oriented systems, but C++ appears to be the language most likely to be widely used. One advantage of C++ is that it permits a more gradual introduction of object-oriented programming techniques since it is a superset of the C language. This can also be a disadvantage, since it allows continuation of conventional C programming techniques even though the language is C++. Since C++ is a strongly typed language, it is more difficult to successfully put reusable parts together. However, there is also less of a burden on the programmer to ensure type compatibility, since the compiler helps with type checking.

The current generation of object-oriented programming environments extend the programming languages with browsing and debugging tools. Some include class libraries which can be used by application developers. ObjectWorks (for Smalltalk and C++) and CenterLine ObjectCenter (formerly Saber C++) both provide additional capabilities which can greatly improve programmer productivity. A few development tools, such as ObjectCraft, permit *visual programming*. These tools offer great promise in applying object-oriented techniques. Such tools can permit developers to specify classes and methods using a high-level approach. These high-level specifications are then used to generate source code. With these front-ends, developers can concentrate on the relationships among objects and leave language-specific details to the code generation facilities of the tool. Future tools might use the state diagrams employed by several OOA/OOD methodologies to generate scripts for controlling object behaviors.

4.4. Software Components - a Strategic Resource

Just as *information* is now recognized as a key part of a corporation's assets, the *software components* created and owned by the organization and the organization's technical skills in using these components are also crucial to the corporation's success and competitiveness. In order to increase software productivity (reducing the average number of hours required to produce a line of code, for example), it is only logical to use the most effective methods and leverage existing software assets (libraries of software life cycle components) to produce new applications with fewer resources. Some of these productivity gains are mentioned in Appendix S. In view of all the benefits of using OOT, a few groups, such as a scientific programming support group at Savannah River (see Appendix T), have made the commitment to use OOT for all new software development unless there is sufficient reason *not* to do so.

Software components are intellectual property, and this raises legal issues and licensing concerns. Implementing software reuse, either commercially or within an organization, will require working through a number of new and interesting questions.

Software reuse is the ability to reuse objects across programs, applications, and development organizations. Reused objects may be code segments, design diagrams, test modules, or any part of the software life cycle which can be shared for later use. Achieving software reuse is the most difficult benefit of OOT to attain, and is the most expensive, since it requires significant time in the design stage of a system to evolve to the right definitions to maximize class reusability.

As the entities, behaviors and relationships understood about the domain are transformed into a class hierarchy, distilling needed functionality into generic classes encourages their reuse. An accompanying problem is that of finding the right object in a library of hundreds or thousands of classes, so that it can be reused.

4.4.1. Use of Class Libraries [3]

Experienced object software developers highly recommend that any group starting to do object work should get a good class library to boost its efforts in building an object system, especially for beginning C++ programmers. There are perhaps a dozen companies offering Smalltalk libraries and this market is mature enough that multiple class libraries can be successfully combined without significant difficulty. The C++ library market is less mature and the library class definitions often clash.

As OOT matures, many class libraries targeted at different software development levels will appear, from the systems or foundation class level up to application-specific levels. (See Figure 4-1.) Some libraries will provide generic functions such as a graphical interface, and will enjoy a high degree of reuse. Other class libraries will be specialized to help different groups of professionals in many disciplines. The lowest degree of class reuse can be expected from the application-specific classes, which are customized for specific uses of models in different situations. Another consideration in selecting class libraries is the target development and application environment, for the libraries are not equally suitable to all situations.

---

[3] This section includes material based on experience described by Chuck Noren of Martin Marietta Defense Systems and Don Herkimer of Martin Marietta Astronautics Group, both in Denver.

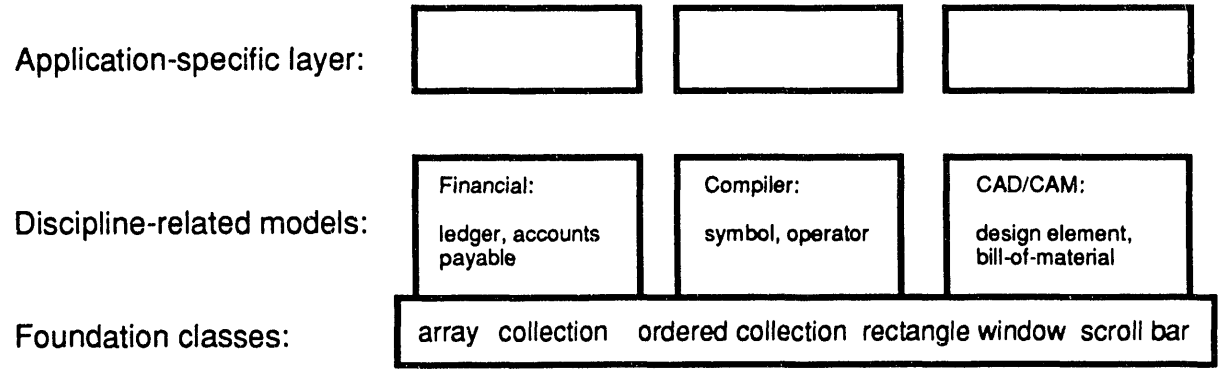


Figure 4-1. Layers of class libraries.

Using existing libraries acquired with source code, a team can save considerable time. The documentation on almost all libraries is not complete enough , and the source can be invaluable, both to understand how the library's classes work, and to emulate the same style and conventions when it is necessary to extend the library.

A number of technical issues must be resolved to enable widespread use of OOT. Fitting multiple class libraries together for use in an application will require careful checking for redundancy or conflicting definitions of objects, and will require conformance to standard interface protocols for communication among the objects. hese standards do not currently exist, but de facto standards are being developed by members of the Object Management Group (see Chapter 6 for discussion of this group's work). Standards efforts are also under way in the American National Standards Institute (ANSI). Also, there are sometimes difficulties when a different version of C++ compiler is used from that of the library supplier.

As an organization matures in its development and use of class libraries, it should have some experts available who are familiar with each commercial or public domain class library which might be useful to its projects. These experts should guide new projects to select the appropriate libraries to use and help train the software team in what is available to them. Public domain libraries are desirable to use, since there are several mature public domain libraries available and this can be a way to provide an open solution to a problem. (X-Windows, while not a class library, is an example of a widely used "public domain" library. The National Institutes of Health Class Library NIHCL, pronounced "nickel", is an example of a mature public domain class library.) Recommended public domain libraries include the NIH Class Library, InterViews from Stanford, ET++, and possibly COOL from Texas Instruments.

It is also good to establish a program of software reuse in each of the major language areas. Every object-oriented developer should become aware of established company policy of how classes are put into the organization's reuse library, and be familiar with and encouraged to write generalized classes. The growing base of reusable software will be a valuable benefit of this practice. Finally, each organization should monitor the developments of the Object Management Group (OMG), which is establishing standards for object-oriented development.

4.5. Risks and Costs of OOT

Any moves to adopt object technology will be driven by the expected benefits and the need to maintain a competitive advantage in software production, but there are also risks and costs associated with such a paradigm shift. Although the Smalltalk language appeared about twenty years ago, object technology as a whole is just beginning to mature. The tools are still not as

robust as more conventional tools and languages. Although the technology promises powerful capabilities in the production of high quality software, the object tools available for use may be lower in performance, at least for the first few years.

One risk of object-oriented software development is that there is often a long period of iterative analysis, design and prototyping of objects before the software components start to appear in demonstratable form. There may appear to be less progress than with a typical procedural development. If a system is prototyped using either approach, much of the user interface and superficial aspects of the system may be demonstrated early on and enthusiastically received. But then the necessary disciplined design and implementation of the production version of the system may go much slower. Most of the time, effort, and responsibility needed in a project are in the analysis and design stages, not in implementation. This can be very worrisome to observers, especially project managers and customers, as they feel that little progress is being made.

One of the big challenges of adopting OOT is getting a body of qualified object-oriented software engineers. Recent graduates of computer science and information sciences programs who enter the Corporation may be already familiar with object technology, information engineering, and other advanced technologies. However, other software developers must be reeducated to be effective in the new paradigm. Both **education** in the object paradigm and the needed shift of mindset and **training** in the needed tools and languages will be required. All software developers must be familiar with the corporate information repository and with whatever libraries of objects (class libraries) are available for their use as components in constructing new applications. It is widely advised to use consultants in the early stages to act as mentors and teachers for the people learning the technology. There will be an initial period of significant cost for any organization to acquire the needed base of software components. After that period, realization of the benefits of object technology will accelerate, for providing new applications becomes a matter of extending the software base to provide the needed increment of capability.

4.6. Organizational and Cultural Changes Needed

Widespread recognition of a corporation's information as a critical asset has already led to organizational impacts such as creation of Chief Information Officers at high management levels in many companies, with software and information models considered part of the information responsibilities of the CIO. Use of object technology to create better business models and better software will serve to reinforce these trends.

Object technology also contributes to the increasing recognition that software development groups' roles are changing. We cannot and should not try to create all the information delivery applications needed in our organizations. Instead, our goal should be to provide high-level tools and environments in which end users can create much of the information and software applications they need to do their jobs. An example of this type of environment might be an executive information system, in which a manager can use a window and icon interface to create an ad hoc report combining some specific information items needed for a special meeting, and can also use a graphical display capability of the system to create hardcopy and transparencies for presentation of the information. Systems such as this are extremely difficult to provide without object technology.

Recalling the ideas mentioned at the beginning of the chapter, our software organizations must change to become less process-oriented and to focus on the software products and environments we create. We must indeed strive to improve our software production process, but we must also take a product-centered view, where the customer's need and the view of the product as seen and used by the customer are paramount.

The evolution to OOT and extensive use of class libraries to develop new software applications will create new categories of jobs and the need for a different reward structure. Mary Loomis of Versant often describes the new job categories when she speaks on object technology. Some people will excel at creating widely useful objects which are efficient yet flexible and easy to reuse. These people will be the *class creators*, and they should be rewarded for the level of reuse their objects receive. Some people will be *librarians*, skilled at (and rewarded for) naming class objects and other software LCOs (life cycle objects) and arranging them so that they can be browsed and found easily when needed for reuse or specialization. *Application developers* must be familiar with the software components available to them and must be skilled at combining these components (unchanged or appropriately customized) to create new applications.

Both consultant Jon Hopkins and Mary Loomis emphasized in an Energy Systems symposium what we have heard many object technology leaders say about changing the reward system for object software development. Instead of rewarding programmers for activity, that is for the amount of code they produce, management must evaluate and reward them for the amount of reuse they do, for that is what will make them more productive.

This switch from creation of original code to the browsing of software catalogs for appropriate items to reuse may lead to some disgruntlement at first. Similar grumblings were heard ten to twenty years ago when engineers trained to design circuits were compelled to browse catalogs and use off-the-shelf components. However, the productivity leaps and new capabilities in the integrated circuit world have been phenomenal, and we can hope for similar improvements in software development. Instead of creating an application from source code, the object-oriented software developer of the future will concentrate on using available tools and components to provide a system that meets the user's known needs, can grow easily to meet probable future needs, and does its job with a flexible and pleasing user interface.

Chapter 5:

Object-Oriented Database Management Systems

An object-oriented database management system (OODBMS) is a database management system which supports an object-oriented data model. An OODBMS must provide persistent storage for the object instances, their behaviors, and their class definitions, or *schemas*. The OODBMS must provide an interface for schema definition and modification, i.e., a data definition language (DDL). The OODBMS must also provide an interface for data manipulation functions, i.e., a data manipulation language (DML) to store, retrieve, and modify the actual objects. OODBMS should also support execution of object behaviors from within the database. This basic functionality can be integrated with traditional database management system facilities (transaction management, recovery, security, integrity, etc.) to provide true object-oriented database management systems.

5.1. Historical Perspective

Numerous data models have been proposed for data management which differ in the manner in which they permit data to be viewed and manipulated. The hierarchical, network, and relational data models have preceded the object model and it is instructive to contrast these earlier data models with the object approach. In the hierarchical data model, record types are linked together in a parent-child relationship, where each parent record type can have multiple children record types, i.e., one-to-many data relationships. The network data model extended the hierarchical data model by allowing record types to have multiple parents and multiple children, i.e., permitting the definition of many-to-many data relationships. Both the hierarchical and network data models implemented data relationships via system-generated embedded pointers that were physically maintained and stored with the actual data. While maintaining such pointer links between record types in a database involved system overhead and made data navigation both complex and limited, DBMS products employing these technologies gained wide marketplace acceptance during the middle and late 1970's.

The newer relational data model discarded the use of system-generated pointer links in favor of using actual data values to permit dynamic navigation and linking operations between data files, or tables. The relational data model was based on set mathematics and set operations. The relational data model significantly simplified the user interface to data definition and data manipulation, although it did not perform nearly as well for some types of applications as the preceding embedded pointer-based data models. By the middle 1980's, DBMS products built around the relational data model had gained command of the general purpose DBMS market, supplanting the predecessor hierarchical and network DBMS products.

At the same time, there were fast emerging new applications requiring extensive manipulation of complex information structures, including CAD data, graphical/image data, text/voice data, geographic/spatial data, etc. These complex data types could not be easily defined using traditional database technology and their restrictive data types, and the resulting overhead of manipulating such complex data typically resulted in unacceptable performance. These types of applications led to the emergence of object database systems.

Prior to the advent of object database systems, storage options for object-oriented programming languages were very limited. Complex object relationships had to be translated into representations suitable for storage in relational database systems or in files, and there was no way to store object

behavior (executable code). The process of reading data back in from the relational structures and reconstructing objects necessitated long setup times. Similarly, at the completion of the program, the objects had to be "flattened" in order to resave the current state of the system. In electrical and mechanical CAD applications, a great deal of time was required for making a single change due to the time spent on the beginning and end of sessions.

5.2. Object-Oriented Database Manifesto

In 1989, a group of academic OODBMS researchers collaborated in writing *The Object-Oriented Database System Manifesto* [ATKI89], a set of "Golden Rules" which must be satisfied for a system to be an object-oriented database system. Here are the rules, without explanation:

The Object-Oriented Golden Rules

1. Thou shalt support complex objects.
2. Thou shalt support object identity.
3. Thou shalt encapsulate thy objects.
4. Thou shalt support types or classes.
5. Thy classes or types shall inherit from their ancestors.
6. Thou shalt not bind prematurely.
7. Thou shalt be computationally complete.
8. Thou shalt be extensible.

The DBMS Golden Rules

9. Thou shalt remember thy data.
10. Thou shalt manage very large databases.
11. Thou shalt accept concurrent users.
12. Thou shalt recover from hardware and software failures.
13. Thou shalt have a simple way of querying data.

The following additional features are proposed to significantly enhance the power and functionality of an OODBMS:

Optional Features: "The Goodies"

1. Multiple inheritance
2. Type checking and type inferencing
3. Distribution
4. Design transactions
5. Versions

The document certainly cannot be considered formal or theoretical, but it does represent a convergence of research opinion on the proper direction for OODBMS technology. In a limited sense, the Manifesto serves the role of Codd's rules for relational databases, in that it establishes design criteria for the next generation of database technology.

5.3. Architecture

Useful characterizations of OODBMS are found in [DABR90, KIM89]. These systems are viewed as consisting of three components: an Object Manager which provides the interface between external processes and the database system, a Transaction Manager which provides client-server capabilities, and the Persistent Storage which is the object representations as actually stored on secondary memory (see Figure 5-1).

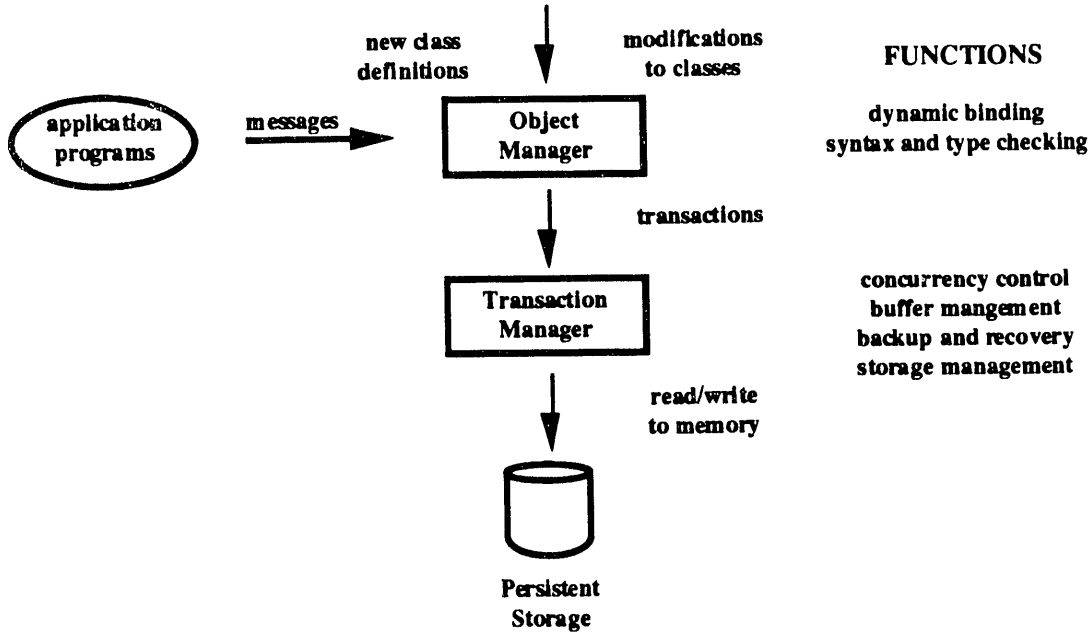


Figure 5-1. Object-Oriented database management.

The Object Manager receives requests to create new class definitions, modify existing definitions, process messages generated by application programs using the OODBMS, and process ad hoc queries. The Object Manager performs dynamic binding and syntax and type checking. Requests are then submitted to the Transaction Manager.

The Transaction Manager (also referred to as an Object Server) manages the actual retrieval, insertion, deletion, and update of stored objects in the database. A single server may handle transactions submitted from more than one Object Manager. The Transaction Manager provides concurrency control, buffer management, and recovery services. It is also responsible for physical storage management which includes both object storage in the Persistent Storage and the implementation of access methods. Typically, backup and archiving services would also be provided by the Transaction Manager.

5.4. Development

The development of OODBMS has proceeded in two divergent directions. In one, object-oriented programming languages have been augmented with many of the services provided by commercial database systems, e.g., persistent storage of objects, archiving facilities, transaction management, and query languages. Such OODBMS are predicated on the belief that it is easier to implement database capabilities into an object programming language than to augment type capabilities in existing database systems. These types of OODBMS may be characterized as "language-centric". While some of these systems may provide excellent performance, they are typically designed for a particular programming language, and therefore may not provide similar performance or even offer support for other programming languages. Also, language-centric OODBMS are less likely to integrate well with other types of database management systems, such as relational ones.

The second common approach to OODBMS development has been to extend or evolve existing relational database management technology to include support for additional data types and query language extensions to accommodate objects. These OODBMS may be characterized as "database-centric", or extensible database management systems [ZDON89]. Extended data types may include text, graphics, voice and image data. A data type for procedures should also be available. Storage of procedural behavior allows databases to become *active* instead of passive. The query language must then be extended to permit selection and retrieval for these new data types. OSQL and SQL3 are object query languages which are described in a later section. Versioning and cooperative processing may also be required. Database-centric OODBMS typically can support a wide range of programming languages.

Of the two approaches to OODBMS development, a larger number of the current marketplace OODBMS offerings have language-centric origins. There are a few database-centric OODBMS products now in the marketplace and even more soon to become available. However, they generally offer lesser degrees of object support "richness" than do the competing language-centric products. The relational database companies expect that they will be able to add some object storage and retrieval techniques to augment their systems. Robert Miner, co-founder of Oracle, was quoted as saying "I was nervous that we were going to be blindsided. But now I think that we'll be able to do everything they do before they do everything we do." [VERI91, p. 100] Both language-centric and database-centric OODBMS approaches have numerous strengths, and both types of OODBMS products are reasonably assured of achieving good marketplace success.

5.5. Performance

Performance of OODBMS versus traditional DBMS (relational, network) is a subject of considerable interest. The caveat to the buyer is that most claims of DBMS performance come either directly from vendors or are vendor sponsored, and vendors invariably can design/tune performance verification suites that portray their products in a favorable way. For complex object data, some OODBMS vendors have documented their products to offer up to a tenfold performance increase over traditional DBMS, and in some cases, this level of performance improvement can likely be achieved. However, OODBMS are still a relatively new commercial technology, and as such, significant performance gains and improvements will in part depend on the wider production use of OODBMS within the marketplace, which in turn will provide OODBMS vendors with true usage statistics and performance data that will serve as a basis for future OODBMS tuning and performance enhancements.

For classic business/MIS database applications with purely standard types of data (fixed alphanumeric, numeric, dates, etc.) and transaction processing requirements, the OODBMS performance levels are generally far less impressive than traditional relational/network DBMS. For example, OODBMS results against the industry standard database benchmarks (e.g., TP/1, debit/credit) yielded lower transaction rates/higher response times when compared with traditional DBMS transaction rates. In summary, traditional DBMS and OODBMS both perform well with the primary types of data they were designed to handle. The major difference at present is that the performance of traditional DBMS has benefitted from more years of experience, research and development, whereas OODBMS are newer technology and so the future is projected to hold great potential for significant performance gains for OODBMS.

5.6. SQL3, OSQL, and OQL(X)

There are active efforts to develop standard database languages for object-oriented database management systems. Database Language SQL is adopted as a standard for relational DBMS by

ANSI (X3.135-1989 and X3.168-1989), the U.S. Government (FIPS 127-1), and ISO (9075-1989). A new upward compatible version of SQL (commonly known as SQL2) is now being concurrently processed through the final procedures within both ANSI and ISO, and is expected to be finalized for formal adoption by both organizations in 1992. At the same time, both ANSI and ISO technical committees are defining the follow-up to SQL2, "SQL3", which will contain extensive new capabilities to accommodate object definition and object manipulation in addition to all previous SQL functionality. Current SQL3 draft specifications include facilities for encapsulation (assertions, triggers), user-defined data types, abstract data types (ADTs), and inheritance. SQL3, like current SQL and SQL2, would offer bindings to multiple host languages.

OSQL (Object SQL) is a high level language for developing object-oriented database applications [LYNG91]. It is described as a combination of the best benefits from object-oriented programming (encapsulation, inheritance, extensibility) with the benefits from traditional database programming (access control, views, concurrency control, declarative queries). OSQL is based on the functional language approach, and provides data access and operations via a set of built-in functions, as well as allowing user-defined types and functions. OSQL is designed to be language independent. Hewlett-Packard implemented a subset of OSQL as part of their Iris Project. They now market Iris under the name OD/1. Interactive as well as programmatic interfaces for OSQL have been prototyped by H-P Labs.

The SQL standard was originally developed to serve both as an interface for interactive users and as a sublanguage to be used by programmers. Experience has revealed that it has been not been fully satisfactory for either use. Interactive users (especially nontechnical ones) generally choose easier query interfaces, such as graphic query or query by example (QBE), and let those interfaces generate the SQL. As for programming interfaces, the DBMS vendors have had to extend the core SQL sublanguage in order to make it more useful for developers. The various extensions then are incompatible with each other, and the interoperability goal of SQL is defeated. Also, SQL is supposed to work with all languages, but its definitions of language constructs often conflict with those of the host language it is being embedded in. This experience with SQL has caused some industry writers to call for a different approach for object data manipulation languages. They propose that each programming language establish a data manipulation language (DML) binding appropriate to that language, which would integrate more smoothly with the host language and take advantage of its strengths, instead of conflicting with them. Such an extension would also be easier for developers to learn [ATWO89]. The American National Standards Institute (ANSI) Object-Oriented Database Task Group (OODBTG) committee recommendations for standards in object information management [MOOR91] include both a query language bound to a language-neutral object model (such as SQL3) and a query language bound seamlessly to different programming languages. This language-specific approach is taken in the OQL(X) efforts, described next.

OQL(X) is an approach to extending a programming language X with a statement from an Object Query Language (OQL) [BLAK90]. OQL(X) is based on three principles:

1. adding a set type to the programming language X,
2. extending the programming language X with a query statement based on the SELECT-FROM-WHERE structure of SQL, and
3. allowing expressions from the programming language X to be used in the formulation of queries.

A prototype binding of OQL with C++ was built as a module of the Zeitgeist Open Object-Oriented DBMS at Texas Instruments. In addition to OQL(C++), a CLOS version, OQL(CLOS), was also built. The result of OQL(C++) is a query facility for object-oriented languages seamless with respect to C++. OQL(X) is proposed as an alternative evolutionary SQL approach for object-oriented database management. It would utilize different SQL extensions for each programming language.

5.7. Legacy Issues

Most existing production database applications reside in network or relational DBMS, and many large business applications still do batch processing with flat file systems. In general across the industry, there are many more network database applications than relational, but that is not true at some places (Martin Marietta Energy Systems, for example). An important challenge is to integrate OODBMS with the "legacy" of applications using these existing DBMS. Before object-oriented technology can be introduced into traditional application shops, there must be some solution to this problem of existing code integration.

OODBMS that are extensions of relational database management systems (database-centric) should theoretically lend themselves well to an evolutionary integration path with existing traditional database applications, at least the relational ones. Proponents of such OODBMS point out that there are no new coexistence/communication issues with this approach, but rather graceful extensibility/migration of existing applications is attainable. They also state that these OODBMS products are "more production quality", which is a commonplace requirement of traditional database applications.

On the other hand, advocates of the language-centric OODBMS are quick to point out that their implementations provide a richer, more pure object-oriented environment, and that traditional database applications can easily coexist and communicate with their OODBMS implementations.

An OODBMS featuring the SQL3 language can solve most legacy issues, as SQL is a standard today for traditional database applications and is being extended to support object-oriented constructs. Another solution is to treat the existing systems as large objects with their own behavior. In this approach, the interface between the object system and the legacy system is treated as a series of interactions between two objects. An object is "wrapped around" the existing system and messages are sent to that object, or "wrapper" (see Figure 5-2).

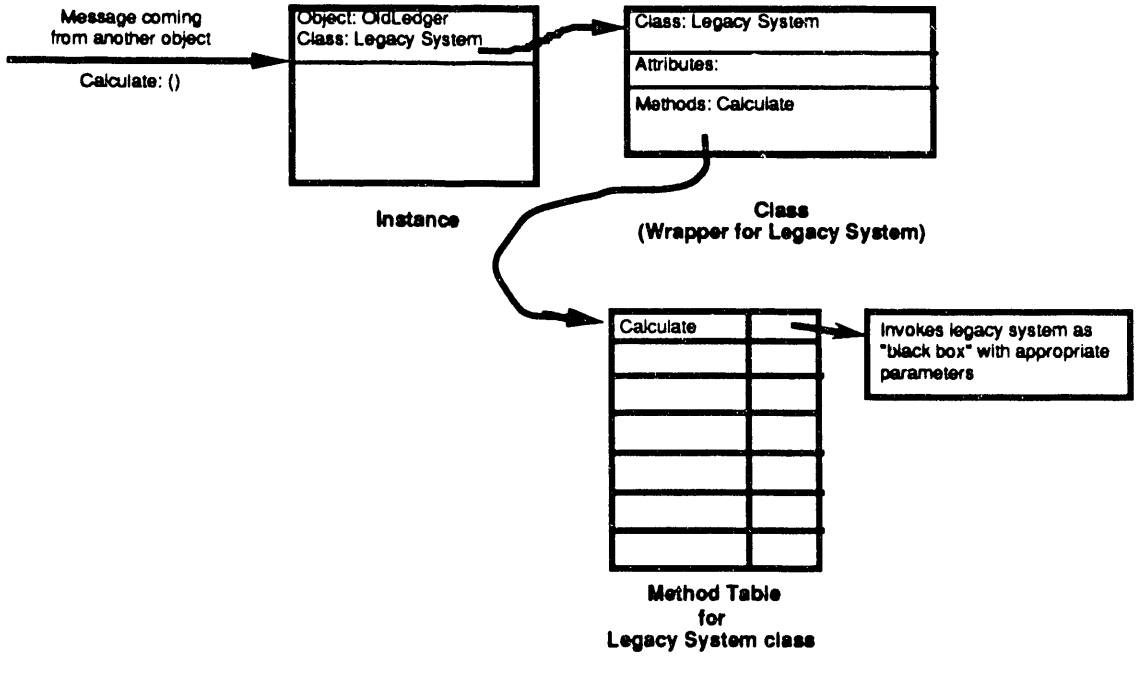


Figure 5-2. "Wrapper" around Legacy System.

5.8. Transparent Data Access

Transparent data access means totally integrated distributed data under the control of a distributed database management system. Distributed database management implies capabilities for partitioning logical data structures into pieces that may be stored at different remote sites, for replicating data at multiple sites with coordinated transaction management, concurrency control, and access management. A truly distributed database is still largely a research consideration. Commercial DBMS products, including OODBMS, are making progress in this direction and many actually offer some distributed database facilities, but usually with many update and concurrency restrictions, and often with severe performance limitations.

What is possible in the near term is distributed processing and interoperability. Distributed processing implies "client/server" access to remote sites, with full capabilities for data definition and data manipulation, and for standardized transfer of data parameters and query responses across communications lines to multiple remote sites. With traditional relational DBMS, SQL is the vehicle for such interoperability. A standard Remote Database Access (RDA) protocol based on a client/server SQL environment is expected to be finalized and adopted by ANSI and ISO in 1992-93. This RDA/SQL standard will serve as a vital building block for future transparent object-oriented database access, for those OODBMS supporting SQL3. Architectural issues for OODBMS include the distribution of objects and the distribution of global schemas into local components. It should be noted that OODBMS lend themselves better to distributed database and client/server architectures than do traditional relational DBMS.

Transparent data access is a strategic goal of most DBMS vendors today. Some selected vendors have joined together in an effort to make their products interoperate and provide easier access to data stored in their databases. ANSI standard SQL is the common denominator that allows such technologies to become marketplace reality. Gateways are a popular method of providing transparent data access. Gateways permit user access to remote sources of data via predefined network routing, while making data appear local to the client user. Gateways can provide concurrent access to data stored at multiple remote nodes, and permit operations to be requested that require data joins, unions, etc. Gateways typically have some restrictions (e.g., may be limited to retrieval only with no update, may be one directional) over full function distributed DBMS capabilities.

Another approach to providing transparent data access is through the use of natural language interfaces. Such interfaces typically allow an English-like query request be specified by a user, and then reference a lexicon-type facility to translate the English request to a valid SQL syntax. This kind of user interface has met with wide acceptance at many sites from end-user personnel, who typically find SQL too complex to master. Such user friendly interfaces do add a layer of translation to a database session, which does require some additional resources and has some potential performance impact. However, any performance impacts of such database interfaces are usually more than offset by increased productivity of the users. ANSI standard SQL is the key vehicle that has enabled software interfaces of this category to generate database queries that can operate against multiple vendor DBMSs.

5.9. Object-Oriented Knowledge Based Systems

Knowledge-based systems (KBS) builders have used object-type knowledge representations for years, because these representations can model natural world objects and abstractions. Most early implementations of this type of structure have been *frames*, which have characteristic sets of attributes (or slots), each of which can have one or more values. Attributes can have constraints on

legal values, and default values can be stored. As with current object systems, frame structures can be inherited to form class structures. A very useful mechanism is the use of *demons*, which are methods or procedures attached to slots or frames, and which are automatically activated whenever the subject slot or frame is accessed or changed (there can be different demons for each case). The pattern matching rules used by KBS tools in effect would do database joins on instances of classes in the knowledge base, thus carrying out a search for stated combinations of values perhaps more effectively than doing similar searches on relational databases. [HARM90b]

The most important difference between KBS frames and true objects is that frames are not encapsulated. Most systems allow rules to directly access the values in each slot. This "feature", which probably results in more efficient search of knowledge bases, must change as OODBMS systems are more widely accepted. However, the demon feature of KBS tools is quite useful for monitoring a system's use and activities, and has been discussed [DABR90] as *active databases*. With active databases, retrieval and update operations can automatically cause the invocation of alerters and triggers, i.e., procedures. This is similar to the "demon" procedural attachments in frame-based systems. Also known as *access-oriented techniques* [STEF86], actions are performed as a byproduct of data manipulation. True object-oriented techniques require messages or some other form of method invocation in order to access data. These active databases appear to violate the current definition of object-oriented encapsulation.

Another blurring of the distinction between object databases and knowledge bases is the use of pattern matching rules to find similar objects in an object hierarchy. Some knowledge-based development tools which support objects use production rules to effect join relations (e.g. Aion's ADS) while other tools use SQL syntax to find patterns in the data (e.g. Level 5 Object). However, such rule-based approaches may not have access to the object hierarchy. In these cases, a rule matches on the members of a class but does not go down to its subclasses.

The ability to define interfaces between objects and to limit an object's view of another object provides the principal distinction between frame-based and object-oriented systems. Therefore, object-oriented programming can be utilized in the same fashion as frame-based systems while providing a greater degree of software assurance.

5.10. Summaries of Some Object Database Systems

The following comments about several OODBMS are not complete representations of the products, but are compiled from vendor literature and comments of colleagues. More formal and organized presentations of information can be found in Appendices E and F and in a report from MIT [AHME91].

Objectivity/DB (Objectivity of Menlo Park, California)

Objectivity/DB is an object database management system designed for engineering applications. It is based on a client/server model. Objects may be linked dynamically to create composite objects or may consist of dynamic arrays called VArrays. Behavior is not stored. The programming language interface supports both C and C++. A graphical browser based on OSF/Motif allows developers to examine the contents of a database.

ObjectStore (Object Design of Burlington, Massachusetts)

ObjectStore uses a proprietary Virtual Memory Mapping Architecture to provide high-performance data management and storage. In ObjectStore, any C or C++ data type can be allocated as persistent. Existing applications can be migrated to ObjectStore using transaction mechanisms. Behavior is not stored.

Ontos (Ontologic of Burlington, Massachusetts)

Ontos is the oldest of the products; it stores class behavior. Ontos does not support part-of relationships, but a class may have an instance of another class as one of its member attributes, i.e. composite objects. Schema modification is restricted to compile-time changes. The structure of a class may be modified at run-time if it has no instances, but data migration is not supported.

Versant (Versant Object Technology of Menlo Park, California)

The Versant Object Database is a multi-client/multi-server distributed database manager that allows objects to be stored, retrieved and updated in a heterogenous computing system. Versant Star provides a consistent set of gateways to integrate existing applications and databases into the Versant information system. The programming languages supported by Versant includes C, C++, and Smalltalk.

Gemstone (Servio Corporation of Alameda, California)

Gemstone is implemented in C and runs on most UNIX workstations. Its Smalltalk interface is graphical, but its interface to C is based on subroutine calls. The data manipulation language of Gemstone is OPAL, an interpreted language which considers all data types to be objects. OPAL does not support multiple inheritance, nor does it explicitly support the notion of composite objects. In OPAL, an instance may be an attribute of another object. Gemstone allows classes and transactions to be created at run-time, and allows class and instance variables to be added or deleted at run-time. Changes in the inheritance hierarchy require recompilation. Data migration of old instances to new types is impossible. Gemstone terminates transactions with storage access conflicts. Gemstone does not provide update notification or version management.

ITASCA (ITASCA Systems of Minneapolis, Minnesota)

ITASCA is an outgrowth of the MCC Orion project. By supporting dynamic schema modification it allows developers and users to modify existing data structures without regenerating the system. ITASCA supports methods and their automatic distribution to clients. When an authorized user updates a method and commits it to the shared database, ITASCA distributes that code to the local sites. Methods then execute on the local machine. Developers may write code in C or Common LISP.

5.11. State of the Art

In a white paper from the National Institute of Standards and Technology [DABR90], the prediction is that object database systems will be used in CAD/CAM, multimedia and complex engineering applications, but that relational database systems and SQL will be used by more conventional applications, e.g., management information systems. The paper speculates that in five to ten years, heterogeneous database environments might emerge which could integrate relational, object, and transaction systems.

Since object database systems are a relatively new concept, there has been little consideration of database security or transaction management. Indeed, the ANSI OODBTG recommendations state that "In Object Information Management systems, there is a major need for more consensus on security, since little work has been done in this area." [MOOR91, p.11]. Vendors have stated that market consideration could influence their efforts in providing secure systems as defined in the new National Computer Security Center guidelines (the "purple book"). One database vendor, Objectivity, claims that it will have a C2 rating for its product soon.

As part of a Martin Marietta Astronautics Group project for NASA, a comparative study of object-oriented database products was produced in November 1990; this document was updated in late 1991 to reflect various OODBMS system upgrades, and is included as Appendix F. See also the listing of database products from the *Object-Oriented Strategies* newsletter (Appendix E) and the MIT study of object database systems [AHME91].

Chapter 6:

Standards

Object technology has enjoyed rapid growth and enormous interest in recent years, and there are a number of successful commercial language and database products to support use of the technology. However, the computing world has become very interested in standards at all levels of computing. The strong drive for open systems comes from the emerging dominance of the client-server architecture (often with heterogeneous systems) and the economic necessity to preserve investments in software and hardware. Users need to mix various hardware and software products on their local area networks and yet have standardized object interchange capability to allow full communication. Consequently, many feel that object-oriented technology cannot be widely adopted and its benefits cannot be fully realized on a broad scale for large users until there are accepted industry standards for languages, class libraries, and databases. Many others warn that premature adoption of standards in a new area such as OOT could inhibit the exploration and innovation needed to develop mature and robust tools. However, standards efforts are under way, and several will be described in this chapter.

6.1. ANSI X3 Object-Oriented Databases Task Group[1]

In January 1989, the Database Systems Study Group (DBSSG) advisory group within ANSI established the Object-Oriented Databases Task Group (OODBTG) "to investigate the subject of Object Database Management (ODM) systems with the objective of determining which, if any, aspects of such systems are suitable, at present, candidates for the development of standards." Note that the OODBTG uses ODM in lieu of OODBMS, and the term "object" rather than "object-oriented". This OODBTG project team completed its work in July 1991, and the public release of the Final Technical Report is expected from ANSI DBSSG by January 1992. Their scope of work included a Survey of ODM Systems, two public workshops on ODM standardization, an ODM Reference Model document, and a Recommendations for Standards in Object Information Management report. The Task Group has now defined and considered standards recommendations for *object information management (OIM)*, which includes the use of objects in programming languages, object models, database management systems, design methodologies, user interfaces, and other related areas. This broader scope results from the group's conclusion that common concepts are used in all these areas, and the development of object versions of all the relevant standards should be coordinated.

The ODM reference model defined by OODBTG provides a framework that is presented as the design space for ODM characteristics (features, capabilities, functions). The reference model provides a comprehensive glossary of ODM terms that builds a common language for discussing ODMs.

---

[1] Much of this information was taken directly from the following documents produced by the OODBTG:

1. Revision R7 of Reference Model for Object Data Management, Craig Thompson, Accredited Standards Committee X3, INFORMATION PROCESSING SYSTEMS, Document No. OODB 89-01R7.
2. X3/SPARC/DBSSG/OODBTG Recommendations For Standards In Object Information Management Revision 7, Editors: Ken Moore, Craig Thompson, and William Kent, Accredited Standards Committee X3, INFORMATION PROCESSING SYSTEMS, Document No. OODB 90-R7.

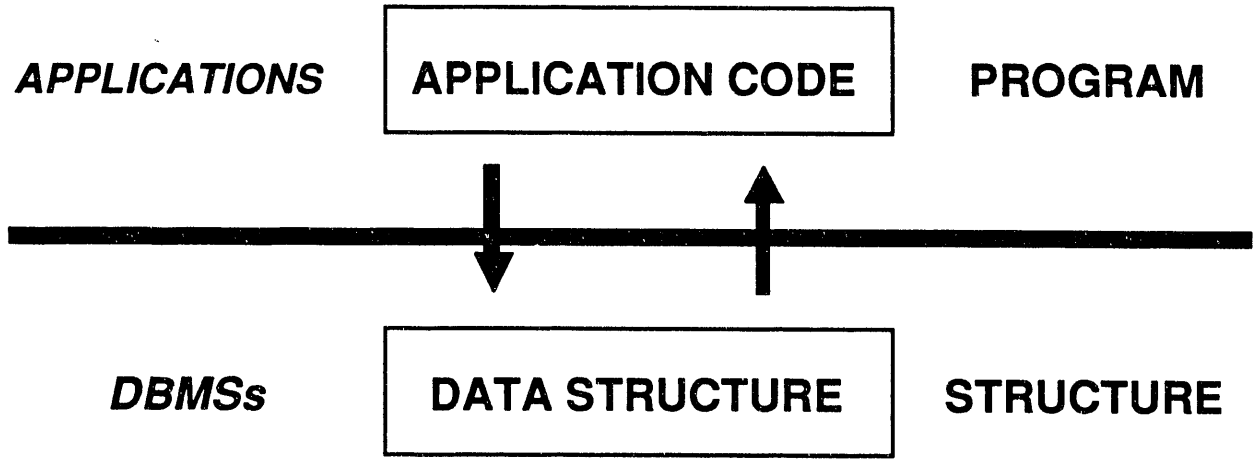


Figure 6-1. Traditional Application/Database Interface.

The OODBTG concluded that today's traditional applications use database operations to manipulate data structures. There is a single interface between user programs and the system code managing the data base as shown in Figure 6-1. The semantics of the operations at this interface are defined in terms of the system-supplied data structures (e.g., records, hierarchies, networks). Object Information Management (OIM) introduces a new middle ground as shown in Figure 6-2. Object-oriented applications do not directly manipulate data structures. Applications apply operations (methods) to objects without knowing how the objects are actually represented or implemented.

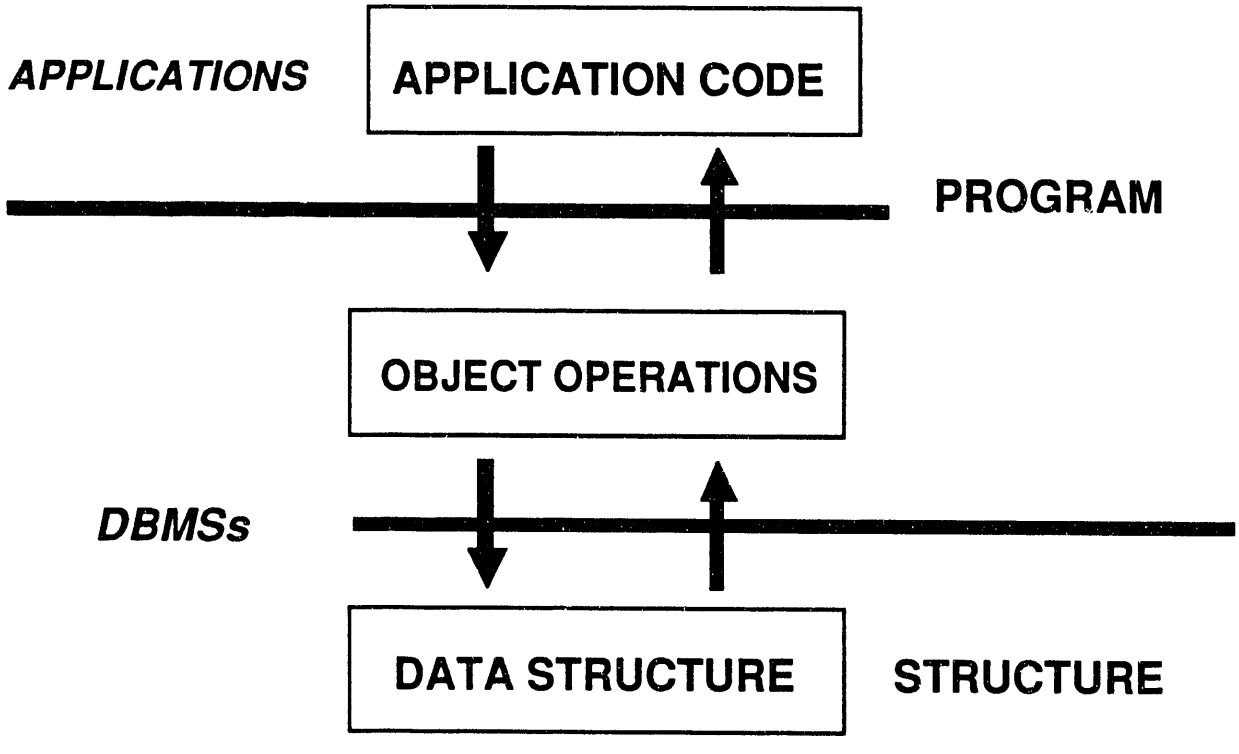


Figure 6-2. Object-Oriented Application/Database Interface

Correspondingly, there are two distinct modes in which information is manipulated. In the first, more traditional mode, data is retrieved from the database to be manipulated by program constructs in the program space. In the second, more objected-oriented mode, operations are executed in the

database space, without exposing the data structure to the requester. Standardization of data structure is more important in the first mode than in the second, which concentrates on object behavior. Existing standards groups organized to follow the traditional boundaries may have trouble adapting to the object-oriented mode. Strong communication among standards groups will be necessary to produce effective, useful standards incorporating the object-oriented paradigm.

The OODBTG classified the related standards into four layers as shown in Figure 6-3. It is their position that interoperation of components of an OIM based on standards requires each standard be coordinated with a common model. The following four levels of coordination were identified:

1. Within a single category, different standards should be consistent.
2. Across a layer, common semantics should apply to similar operations, but no redundant operations should be provided.
3. An integration framework needs to be coordinated with the concepts of a specific application domain.
4. Object model concepts and terminology should be defined and implemented in the same manner throughout all standards.

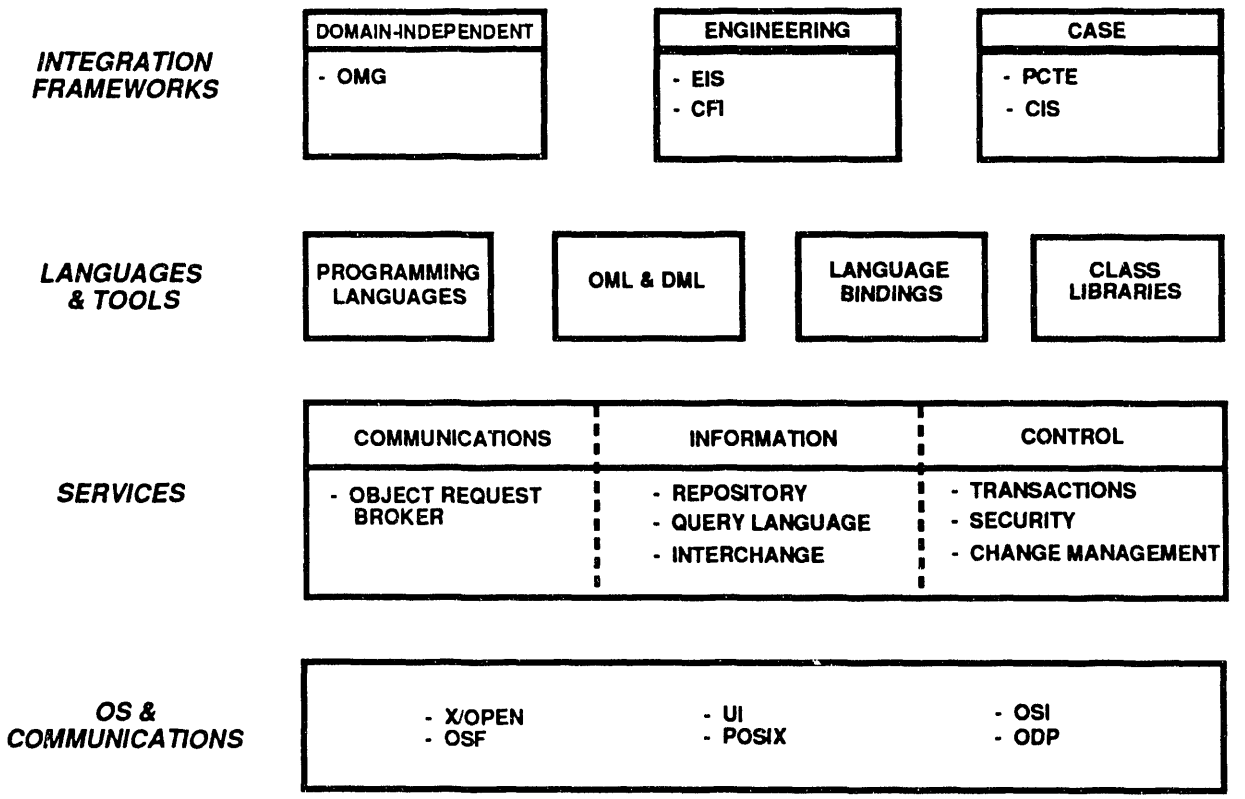


Figure 6-3. Layers of Object Standards

The OODBTG concluded that several OIM areas needed immediate formal standardization attention while others areas were lower priority and could defer standardization for the present time. Subareas of OIM where standards should be developed include:

1. Object Model - a common, unified object model definition, semantics and syntax are needed to facilitate sharing objects across language, operating system and machine boundaries.
2. Object Communications - both ANSI X3T3 (Open Distributed Processing) and an Object Management Group task force are already working on this specification.

3. Persistence - a language-neutral model of persistence, with sections on various languages like C++, Common Lisp, Ada, SQL, etc.
4. Transactions - new transaction models are needed due to new application domains. This area must be compatible with existing or in-progress transaction standards from ANSI X3T5 and X/Open.
5. Query Capability - next-generation database applications need persistent languages and object query languages like SQL3. ANSI X3H2 is developing SQL3.
6. Object Repository - to provide shared definitions among OIM components. ANSI X3H4 is currently developing this.
7. Class Libraries - common contents and organizations of sharable and interoperable class libraries within and across languages.
8. Object Design Methodologies - add methodological considerations to current perspectives on using object technology.

6.2. Drive for Standards

Though there are a number of increasingly mature products on the market, there is little interoperability between the systems, and no common object model has been developed. This lack of agreement on a standard creates a risk for users, since their increasingly complex applications require capabilities beyond those provided by conventional systems. Driven by these needs to use object tools from a nonstandardized marketplace, they risk using a product which could ultimately diverge from the de facto standard and could as a result go out of business, leaving the users unsupported.

Strong communication among standards groups will be necessary to produce effective, useful standards incorporating the object paradigm.

Within ANSI, there is coordination in the way standards are developed by the database committee, language committees, repository committee, and similar standards bodies. From Figure 6-3, it is possible to identify other existing standards bodies that have been or will be influenced by object-oriented technology. Strong liaison between ANSI X3 subgroups and other standards bodies will be required, as one of the major challenges will be to ensure that the standards developed are interoperable and useful when assembled together into an OIM environment. Several of the standards groups have now embraced the object paradigm as central to their work. Table 6-1 describes the existing standards groups and addresses their relationship to object-oriented technology. One important consortium, the Object Management Group, is described in the next section.

OOT and OODBMS are such hot topics that various groups are driving hard toward particular areas of standardization. Perhaps the major challenge for OOT standardization is not the technical aspects, but instead is found in the political arenas of standards development. For example, there are diverse groups and bodies who are currently at various organizational and procedural stages of activity for development of a standard object-oriented database query language. Within ANSI, the charter for such a standard will reside with the X3H2 Database technical subcommittee, and will be developed as part of SQL3. However, the current X3H2 membership is comprised mainly of traditional DBMS vendors and users; there needs to be greater active involvement from some of the OODBMS players (Versant, Servio, etc.) in this standards arena. Other standards bodies looking at object query languages include IEEE and some industry specific trade groups. Again, the major challenge and opportunity will be to ensure that OOT standardization efforts are coordinated among standardization bodies/groups. Within ANSI, coordination is an inherent part of the standards process, but in some other arenas, the coordination will require greater voluntary efforts on the part of the players. Without formal coordination of OOT standardization efforts, the result will likely

be that OOT technology and marketplace usefulness will suffer from fragmented and disjoint standards and the associated politics.

Table 6-1a[2]. Standards efforts which relate to object technology.

Standards Effort	Description
	Database Management
X3H2 - SQL3	A technical committee responsible for the standardization of database languages NDL and SQL. They have, in May 1991, completed specification of SQL2, and are currently working on SQL3, an extension to current SQL standard which will include object concepts.
X3H2.1 - RDA (Remote Data Access)	A task group under X3H2 on Remote Data Access (RDA). This group is responsible for the specification of a protocol concerned with providing access to data stored at remote sites using SQL.
JTC1 SC21/WG3 - Data Management	An international standards committee responsible for the specification of standards on data management. Projects include data management reference model, database languages SQL, IRDS and RDA.
SQL Access Group	A consortium of users and vendors working to advance the RDA protocol and planning to work on a call-level interface to SQL systems.
	Transaction Processing
X3T5 - TP (Transaction Processing)	A task group under X3T5 (OSI) is responsible for the specification of TP which is an application layer protocol used for exchange of information between two or more distributed systems.
JTC1/SC21/WG5	An international standards committee responsible for the specification of standards on transaction processing languages and bindings, including concurrency, commitment, and recovery (CCR).
POSIX 1003.1	A group working on a profile for transaction processing.
X/Open Transaction Processing	A working group developing the XTP model of transaction processing, which includes the XA transaction specification.

[2] Source: Elizabeth N. Fong et al., "X3/SPARC/DBSSG/OODBTG Final Report," from *Accredited Standards Committee X3, Information Processing Systems*, September 17, 1991. Used with permission of Elizabeth N. Fong, National Institute of Standards and Technology.

Table 6-1b. Standards efforts which relate to object technology.

Standards Effort	Description
Object Communications and Distribution	
X3T5 - OSI (Open Systems Interconnection)	A technical committee responsible for the specification of protocol standards in accordance with the 7-layer Open System Reference Model. In particular, the X3T5.4 Network Management Task Group is responsible for the specification of managed objects using object-oriented technology.
X3T3 - ODP (Open Distributed Processing)	A U.S. technical committee contributing to the international effort JTC1/SC21/WG7. The ODP effort is working on the specification of a standard reference model which will make the complexities of distributed computer systems more transparent. The ODP-RM defines an ODP trader which is a computational object offering services to other objects at service ports.
OMG ORB-Object Management Group's Object Request Broker	A task force within OMG developing technology that performs application invocation and and sharing of large granule objects.
JTC1 SC21/WG4 - Management Information Services	An international standards committee responsible for the definition of the information model of managed objects that corresponds to the information aspects of the systems management model. Although the documents refer to CCITT applications, they define general object management concepts.
X3T1M1.5	A technical committee responsible for common management information services for managed objects defined in accordance with JTC1 SC21/WG4 documents.
OSI/NM Forum	An international forum on OSI network management.
Data Interchange	
X3T2 - Conceptual Schema for Data Interchange	A project under X3T2 working on the standardization of conceptual schema mechanism for data interchange. Responsible for ASN.1, a language for data encoding and interchange.

Table 6-1c. Standards efforts which relate to object technology.

Standards Effort	Description
Domain-specific Data Representations	
PDES/STEP (Product Data Exchange using STEP)	The PDES is the U.S. organizational activity that supports the development and implementation of STEP. STEP is the standard for the exchange of product model data. The level 3 product data sharing implementation specifies that multiple user applications access data to a common shared database.
EDI (Electronic Data Interchange)	EDI is an application layer protocol. It is a standard which describes formats for orders, payments, shipments, billings, and other business transactions.
EDIF (Electronic Data Interchange Format)	A format for exchanging CAD chip design data.
ODA (Office Document Architecture)	ODA is a standard for interchange of documents (including text, facsimile and graphics information) which are produced in an office environment. Interchange of ODA documents may be by means of data communications or exchange of storage media.
Repositories	
X3H4 - IRDS (Information Resource Dictionary Systems)	A technical committee responsible for the specification of IRDS1 family of standards. This IRDS1 family of standards includes a command language and panel interface specification, a soon to be approved Export/Import File Format standard, and a Service Interface specification. The next family of IRDS standards will utilize object technology.
X3H6 - CIS (CASE Integration Services)	A technical committee working on a family of standard interfaces between CASE environment framework components and tools. Standards are being developed for service and tool registration, change management (versions and configurations), and an object model.
EIA CDIF (Electronic Industry Association CASE Data Interchange Format)	An industry association established to permit interchange of CASE models and data among tools.

Table 6-1d. Standards efforts which relate to object technology.

Standards Effort	Description
	Programming Languages
X3J4 - COBOL (OO COBOL)	A technical committee responsible for the standardization of the COBOL programming language is working on extensions to COBOL that will include object concepts.
X3J9 - Pascal	A technical committee working on the standardization of the Pascal programming language, which is working on a Technical Report for object-oriented extensions to Pascal.
X3J13 - Common LISP	A technical committee working on the standardization of Common LISP which includes the Common LISP Object System (CLOS).
X3J16 - C++	A technical committee responsible for the standardization of C++ programming language.
X11/SC1/TG11 - MUMPS	A task group working on object-oriented extensions to MUMPS
Smalltalk, Objective-C, Eiffel, ...	Object programming languages.
Ada Joint Program Office	The group coordinating the development of Ada 9X.
	Frameworks and Consortia
CFI - CAD Framework Initiative	CFI is a consortium chartered to define interface standards that facilitate integration of design automation tools for the benefit of end users and vendors worldwide.
OMG - Object Management Group	The OMG is a consortium to promote object-oriented concepts and methods. The OMG architecture defines an interface called Object Request Broker (ORB). The Object Model Task Force is developing a description of a concrete object model.
PCTE - Portable Common Tools Environment	PCTE is an emerging ECMA standard for specifying interfaces which are primarily designed to facilitate communications and interoperability among cooperating CASE tools and applications.
OSF - Open Systems Foundation	A consortium which, as part of its project, is defining a distributed management environment (DME) which utilizes object concepts.
ESPRIT	Esprit is a European funding agency working on the specification of information system architectures, including CIM-OSA, COMANDOS, CSA, AND DELTA-4.

Table 6-1e. Standards efforts which relate to object technology.

Standards Effort	Description
	Frameworks and Consortia
EIS - Engineering Information Systems	A US Air Force sponsored framework effort.
X/Open	A consortium of users, hardware and software vendors, developing portability guides for languages, databases, and operating systems.
DARPA	The US DoD funding agency funding work on: knowledge representation Standards Initiative, Open OODB, and NIST Persistent Object Testbed.

As this report approached publication in May 1992, the ANSI X3 Committee announced formation of a new Technical Committee, X3H7 -- Object Information Management.[3] X3H7 will develop a reference model technical report, with scope including:

- an interoperable object model
- object data management services
- external representations of object model schema and data
- object class libraries
- object languages
- object communication and distribution
- object design and methodologies.

6.3. Object Management Group

In the interim period before a formal standard or family of standards exists, a large group of more than 140 commercial enterprises and user organizations formed in 1989 a consortium called the Object Management Group (OMG). OMG "will create industry standards for commercially available object-oriented systems by focusing on Remote Network Object Access, Encapsulation of existing applications, and Object Database Interfaces" (quoting their brochure).

The first order of business for the consortium was the creation of a framework of the basic components of object technology that OMG would like to standardize. OMG does not concern itself with specific implementation of object techniques, but does seek agreement on interface issues of *interoperability*, to assure that message passing between objects in different languages, tools, databases and operating systems can be carried out smoothly. Figure 6-4 from the OMG Architecture Guide shows the major interfaces OMG is considering. The OMG architecture has four major components:[4]

1. Object Request Broker (ORB) enables objects to make and receive requests and responses.

---

[3] For further information on the X3H7 Committee and its meetings, contact Elizabeth Fong, N.I.S.T., Technology Bldg, A266, Gaithersburg, MD 20899, phone 301: 975-3250.

[4] Taken from Object Management Group Standards Manual, Draft 0.1, OMG TC Document 90.5.4, by R.M. Soley, May 25, 1990.

2. Object Services (OS) is a grouping of services with class interfaces that provide basic functions for realizing and maintaining objects.
3. Common Facilities (CF) is a group of classes that provide general purpose capabilities useful in many applications.
4. Application Objects (AO) represents the collection of classes that are specific to particular end-user applications.

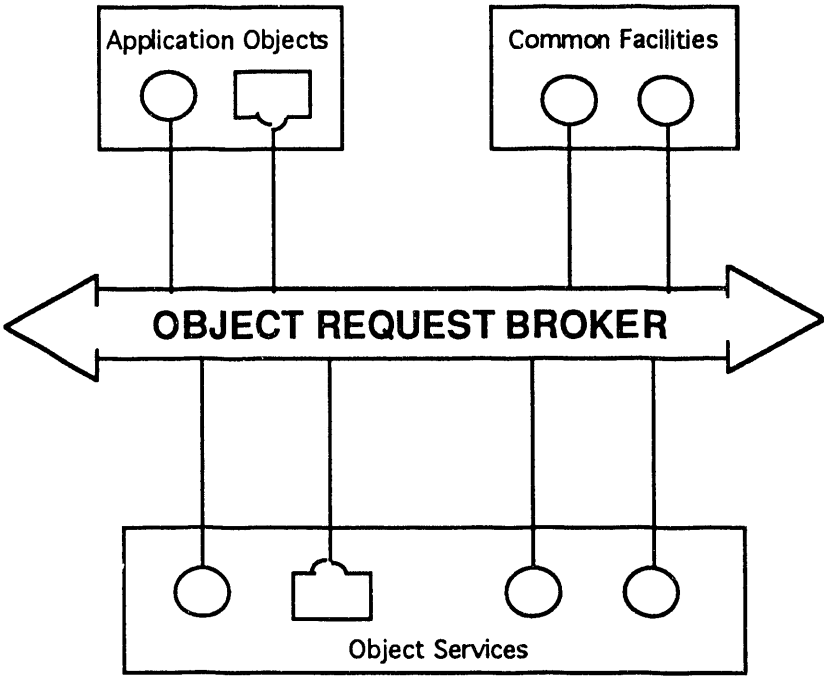


Figure 6-4. OMG Framework overview.

Not surprisingly, the first component area for which a consensus design was attempted was the ORB (see Figure 6-5). Out of an original seven proposals, several were combined and finally there were two: joint proposals by DEC/Hyperdesk and by Hewlett-Packard/Sun/NCR. In an impressive feat of consensus-making, the OMG task force issued an ultimatum in June 1991 and got agreement of the two teams to work out a compromise design for the ORB in the next ninety days. All parties seemed to realize the benefits of working out a combined design, so future cooperation of this sort may be expected on later issues.

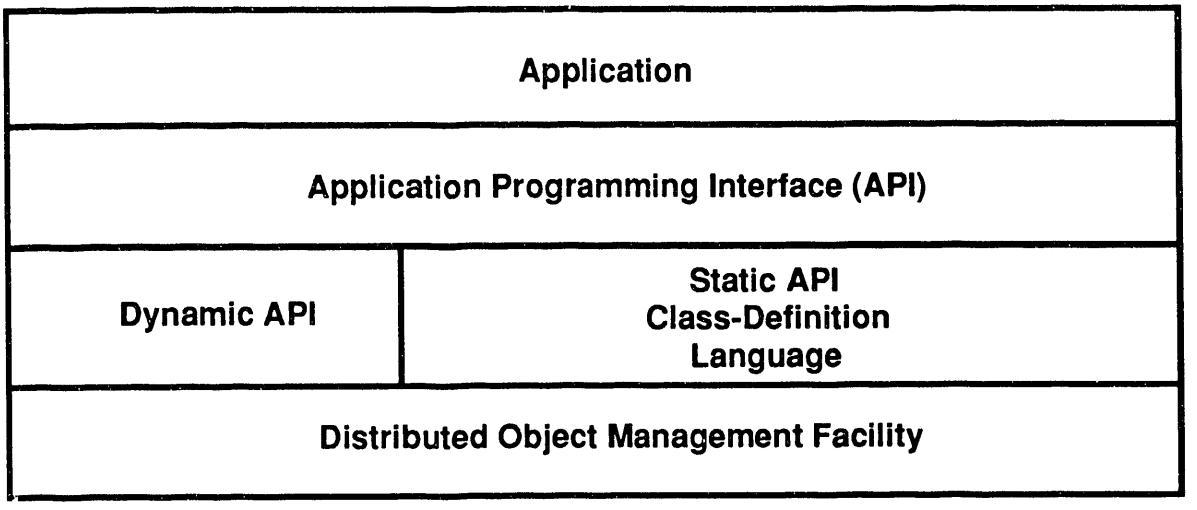


Figure 6-5. Object Request Broker interfaces.

There are some important things to note about the OMG. Instead of waiting for the formalized standards-setting-with-public-review-process, which could take a couple of years, the group has focused on existing technology that can already be delivered. As soon as the agreements are reached, the marketplace can start aligning their products with them. Also, virtually all of the major hardware and software vendors active in object technology are now members. After holding out for over a year, finally even IBM and Microsoft joined OMG in 1991. Finally, in late spring of 1991, OMG and the technical committee of Unix International announced that they will cooperate with each other, and that UI will include key components of OMG's specifications in Roadmap, the future direction of Unix System V. All this is very encouraging evidence that the object tools market may avoid the confusing state of having multiple, semi-compatible products with which to work.

6.4. Status of Standardization in Several Areas

The language Smalltalk is probably the most uniform of the object programming languages, with syntax, semantics and class libraries being fairly consistent between the various dialects of the language. There are three commercial dialects (Smalltalk-80, Smalltalk/V, and Tektronix), plus several public domain versions to be standardized. Issues include run-time representations, compilers, exception handling, and class hierarchies. Smalltalk has the most mature libraries, applications, and toolkits among OOPLs.

The C++ language is undergoing standardization within ANSI by the X3J16 technical committee. C++ is rapidly becoming the de facto industry standard in the United States. C++ is more desirable to industry than Smalltalk because of the easier transition from other languages, such as C. In addition, C++ is not a "closed" language like Smalltalk is; C++ modules can be embedded in other applications. The original ANSI X3J16 scope of work included three layers of C++ standardization: 1) the features and libraries available in the AT&T C++ Release 2.0, 2) primary libraries, support environment features, and run-time mechanisms, and 3) new language features. ANSI X3J16 first convened in December 1989, and projects completion of the C++ standard in 1992. Meanwhile, there are several compilers available and none of them interoperate. This of course causes problems with projects which use tools based on different vendor's language products, or even different releases of the same product, but this long-experienced type of problem is certainly not limited to object-oriented products.

In the area of object-oriented databases, there is no formal, common object model such as E.F. Codd's definition of the relational data model. The nearest such definition is "The Object-Oriented Database System Manifesto" [ATKI89]. The market is still developing, with several good products using different approaches. As yet there is no consensus on a clear leader among the products. However, we can expect progress in this area, given the widespread cooperation the Object Management Group is getting in its efforts to develop de facto standards.

As for analysis and design, there are several methodologies and no clear winners there, either. Some canonical methods of representing applications graphically will probably emerge in a few years. Given the marketplace, it is likely that one of the big CASE vendors will become the de facto standard with added object features. The successful CASE product of the future will probably be UNIX-based and will embody a well-accepted methodology.

6.5. The Impact of Object-Oriented Technology on Software Standards

There are a number of software architecture and development standards in place around the DOE complex. Most assume a structured analysis and design type of software methodology, but for the most part, use of object technology does not appear to conflict with the standards. Some mention object technology in passing, but do not address it further. Methodologies with detailed step-by-step procedures (such as Information Engineering) may not fit the object paradigm currently; however, those methodologies are expected to adapt to object technology over time.

One area in which object databases probably could not meet regulations is in secure or sensitive environments, since there are essentially no security features in the existing products at this time. However, Objectivity claims that they will have a C2 rating for their product soon.

6.6. A Unifying Paradigm

One of the philosophically pleasing aspects of object technology is the unifying influence it has on the way we view computing and software systems at all levels. As the ANSI OODB Task Group points out, object information management is no longer as distinguishable from other aspects of information processing as it used to be. By encapsulating data structures behind object interface protocols, the object model defines the interfaces in terms of domain-specific operations, not as structures access by system- or structure-defined operations. Quoting from the Recommendations for Standards [MOOR91, p. 5]:

> "The semantics of object technology cross traditional boundaries, being applicable to user interfaces, programming languages, network management, repositories, operating system services, storage management, and other areas. It should be possible, but not required, to have uniform semantics, syntax, or both for transient data (typically associated with a programming language) and persistent data (typically associated with a DBMS). Mechanisms developed for distributed object management should also be useful for object data management. ...
>
> "Thus a *very strong* recommendation is to re-examine traditional boundaries, both in computer technology and in standards organizations. Database may no longer be an isolatable component of a software system. Relevant standards should be developed cooperatively by database committees, programming language committees, repository committees, network management committees, and so on. Object technology standards should be developed in a way that promotes harmony across these boundaries."

This view of object technology could allow us to place all tasks related to computing along a continuum, instead of viewing different groups' task areas as distinct and largely unrelated. Adopting this view could have far reaching effects on our organizations and our approach to our work over a period of years.

A quote from Cliff Reeves, IBM's executive responsible for object technology will conclude this chapter:[5]

> "The nice thing about OO is the way you think about objects scales all the way from implementation issues, like binary flow and integer collections ... up to very abstract enterprise-level things like hotel, process, customer, accounts payable, certificate of deposit. You think about these things as nicely capsulated [sic] objects which have certain behavior. I think that's the underlying strength of the model."

---

[5] The DataTrends Report on DEC and IBM, May 1991, p. 2

Chapter 7:

Future Developments in Object Technology

7.1. Changes in Computing

In order to look at the future of object technology, we must consider what is happening in general in the computing world. In 1991, virtually all computing activity is still procedural programming, doing numerical or data manipulation. Most developers and many users use 16-bit PC workstations, but user interfaces must still be aimed at "dumb" terminals hooked to mainframe computers. Most applications use data stored in relational or hierarchical databases. Our computing organizations are still arranged as they were when mainframe applications were the rule, and people trained in the mysteries of programming each specific hardware system provided all applications and preplanned data access. There are indeed many people today using 32-bit hardware and advanced software engineering methodologies such as information engineering to develop software. There are a number of groups using object and knowledge-based techniques and exploring object databases. But these people are part of the first wave of exploiting the new technologies; their work cannot be called mainstream activities.

Fundamental changes will come as several technologies mature. The growth in use of 32-bit workstations is leading us to view computing in terms of networks of clients and servers communicating in a network, instead of terminals talking to mainframes. The field has finally matured enough to commit to open systems, allowing more transfer of software and skills among different hardware systems. There are also other trends to the decentralization of knowledge, responsibility and power in computing; there is a growing trend to provide automated help systems and on-line reference material with hardware and software systems, so that users can independently learn to use the systems and can help themselves in case of problems. More types of programming environments are appearing, too, not just CASE tools for professional programmers, but user-friendly tools which help users to build the types of data access, manipulation, and reporting functions they need (without waiting for the long process of formal application development).

The CASE tools and associated methodologies for professional programmers will help the current large application development situation immensely. Using these emerging tools and methodologies, application development becomes less an art in coding with a programming language, and more a matter of engineering, where the product is analyzed and developed according to a disciplined process. As the tools and our skills in using them mature, we will be better able to estimate and adhere to schedules and resource allocations, and needed documentation will be automatically assisted.

By the end of this decade, these trends will have given us a new world of computing. Most developers and many users will be using 32-bit workstations arranged in client-server networks. Software will routinely be engineered with CASE tools, but these tools will use object-oriented techniques plus knowledge-based and other artificial intelligence techniques to provide much more intelligent, helpful, and easy-to-use applications. Large applications will still be developed by traditional computing application departments, but many applications will be built and maintained within user organizations by means of powerful, WIMP-interfaced programming environments. These environments will run on local workstations, but will build applications which can access (or provide) information from (or to) network nodes all over the enterprise, by means of transparent data access. There will be no reason for concern about what type of hardware is used at another node of the network, for the open systems can communicate using standard protocols.

To make all this happen, there will be extensive use of object technology, because it provides mechanisms for dealing with all the heterogeneous pieces of the computing enterprise. Mainframes and PCs can also be included in the client-server network, and old software systems can function as objects in themselves, interfacing with newer object systems according to standard protocols. The tremendous investment in conventionally represented information and software (the "legacy") cannot be discarded, but it can be augmented with object databases and object-oriented systems where the need and technology are appropriate.

Some may think this view utopian, but they should also look back ten years and consider the differences between mainstream computing in 1981 and the current situation. There are also a number of articles which predict these revolutionary changes [HARM91d], [THOMA91]. Let us next consider how object technology will contribute to these changes.

7.2. The Technology Life Cycle

The lifetime of any computing software or hardware progresses through stages:

- research in academic or corporate settings
- increasing interest and offering of early commercial systems, often by academics starting small businesses
- early adoption by progressive organizations; pilot projects by many large organizations
- appearance of numerous commercial products
- adoption by mainstream organizations
- shakeout of weaker commercial products
- widespread use of mature technology
- declining use as old technology replaced

Figure 7-1 [HARM91b] shows how *Intelligent Software Strategies* editor Paul Harmon views this life cycle, and where he would place a number of technologies. Object-oriented programming languages such as Smalltalk and C++ have made fairly significant progress along the path, both having a number of successful applications produced by large organizations. Really widespread adoption and use of these languages depends on OOT education and language training, plus on the availability and acceptance of class libraries, so that users can readily assemble large applications. Class libraries will become a significant part of the market, whether as standalone products or as elements of object application tools. Also needed are supporting software tools such as class browsers and debuggers to further speed up development times for some languages. Given these developments, many software gurus are predicting that object-oriented programming will be the dominant mode, surpassing procedural languages, within ten years. Indeed, Yourdon and Constantine predict that most professional programmers will prefer object programming by 1995, but that seems a bit early.

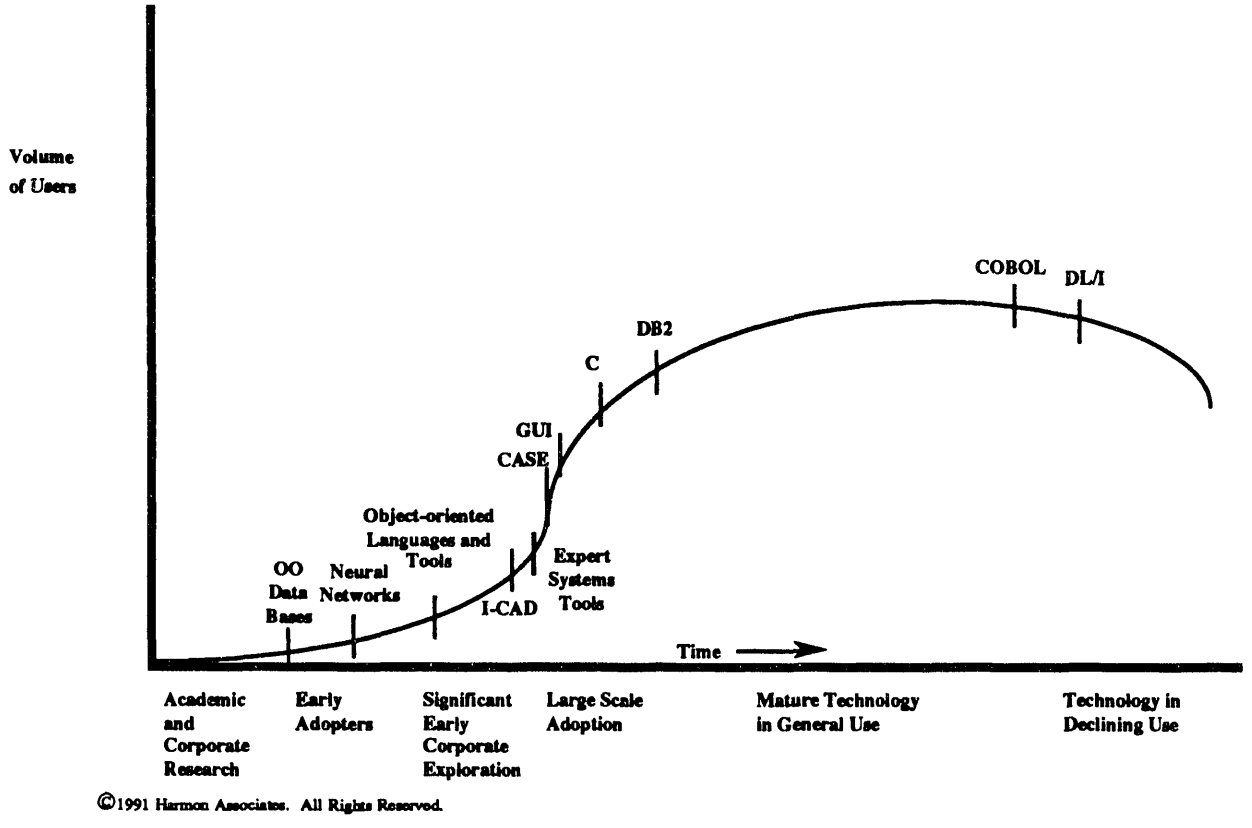


Figure 7-1. The location of various technologies in the software life cycle as of 1991.

If the experience with fields such as knowledge-based systems is any guide, what will be widely used in several years is not the object programming languages themselves but application development **tools** built from those languages, with all the helpful features rolled into a commercial package. An early example of this is ObjectWorks from ParcPlace, which has both Smalltalk-80 and C++ versions. An example of evolution from another direction is that Intellicorp, originally a leading knowledge-based systems tool (KEE) vendor, has enhanced the object-oriented portion of its tool and has entered the object market with strong object-oriented tools (ProKAPPA, KAPPA PC). In addition, the conventional CASE vendors are using object capabilities to implement their tools, though none of the major ones offer the ability to do object-oriented development with the tools yet. Most object development tools currently are from small start-up companies, basing their products on Smalltalk or C++. Some of these will grow into Intelligent CASE (ICASE; note that this acronym is often used for Integrated CASE also) environments themselves. We can also expect to see some companies merge with or acquire others to gain advantageous technology. As the market develops, versions of products will be ported to more hardware platforms. It is impossible to say which vendors' tools will dominate by the late 1990's, but it seems certain that the successful ICASE tools of that time will be an amalgam of the best features of today's CASE, KBS and OO tools.

Object-oriented database products are not as far along the life cycle as languages and development tools are. There are fewer than ten major products, and all are standalone products for workstations. The OODBMSs are already valuable for CAD/CAM applications, enabling persistent storage of the complex knowledge structures needed for those applications. The competition in the database area will be between the OODBMS vendors who will be expanding their capabilities, hardware platforms, and performance, and the relational database vendors, who

will be modifying and extending their products to add object capabilities. If the relational vendors can successfully add enough object extensions fast enough, they are likely to win the market, given the cost and trouble of reimplementing any organization's databases and the urge to minimize any needed conversions. However, many legacy systems are file-based (not relational database systems) and running on mainframes. If a vendor can show a good migration path to workstations and OODBMS, that might be a winning strategy in many cases.

To summarize, in ten years, the most successful **application development tools** (or Intelligent CASE products) will combine the best features of today's CASE tools, knowledge-based systems and object-oriented tools. It's also expected that there will be an **object-oriented interface and operating system** that will be easy for nonprogrammers to use and modify; the NeXT machine interface might be an early example of this. And both these types of systems will be integrated with **object-oriented databases** which will store and manage the complex structures of multimedia needed by the applications of that day (see Figure 7-2). [HARM91e]

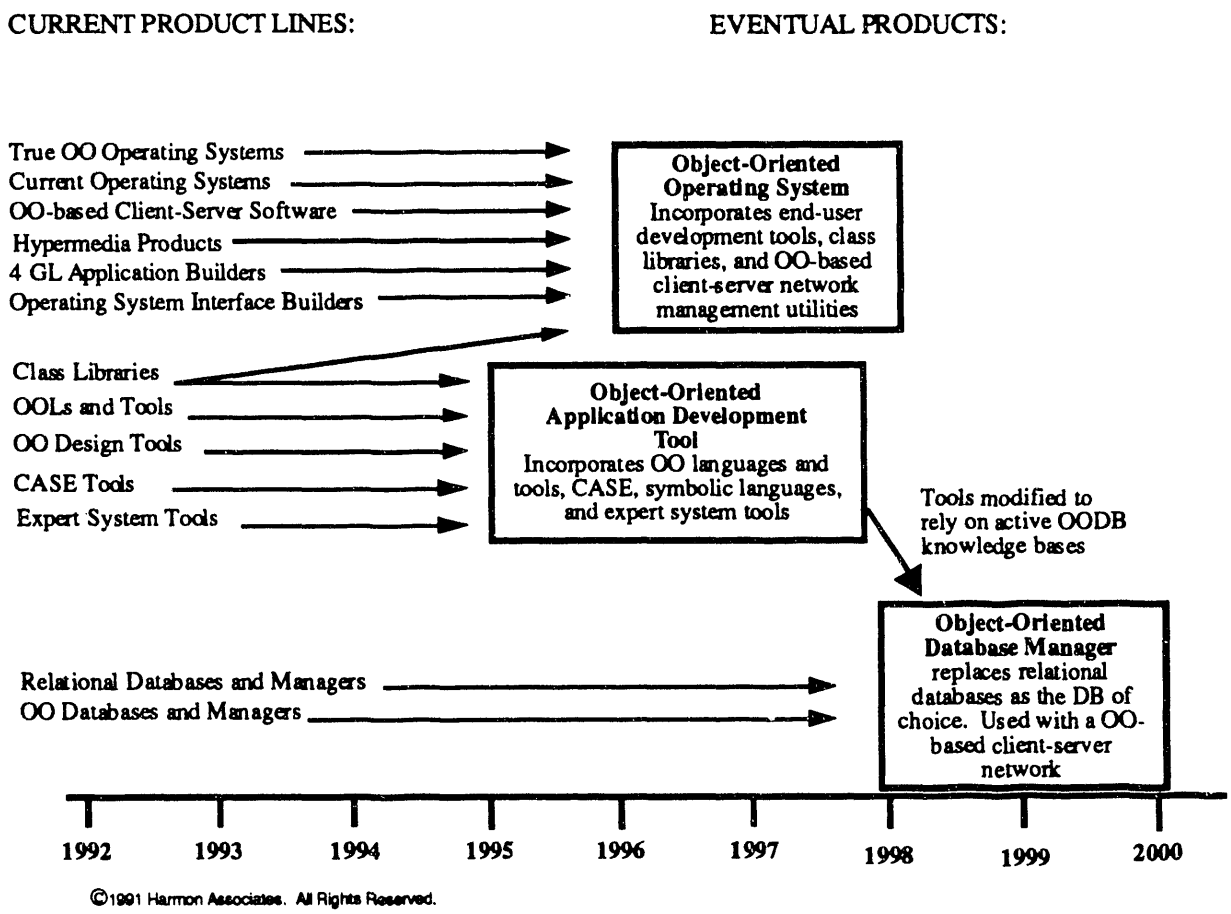
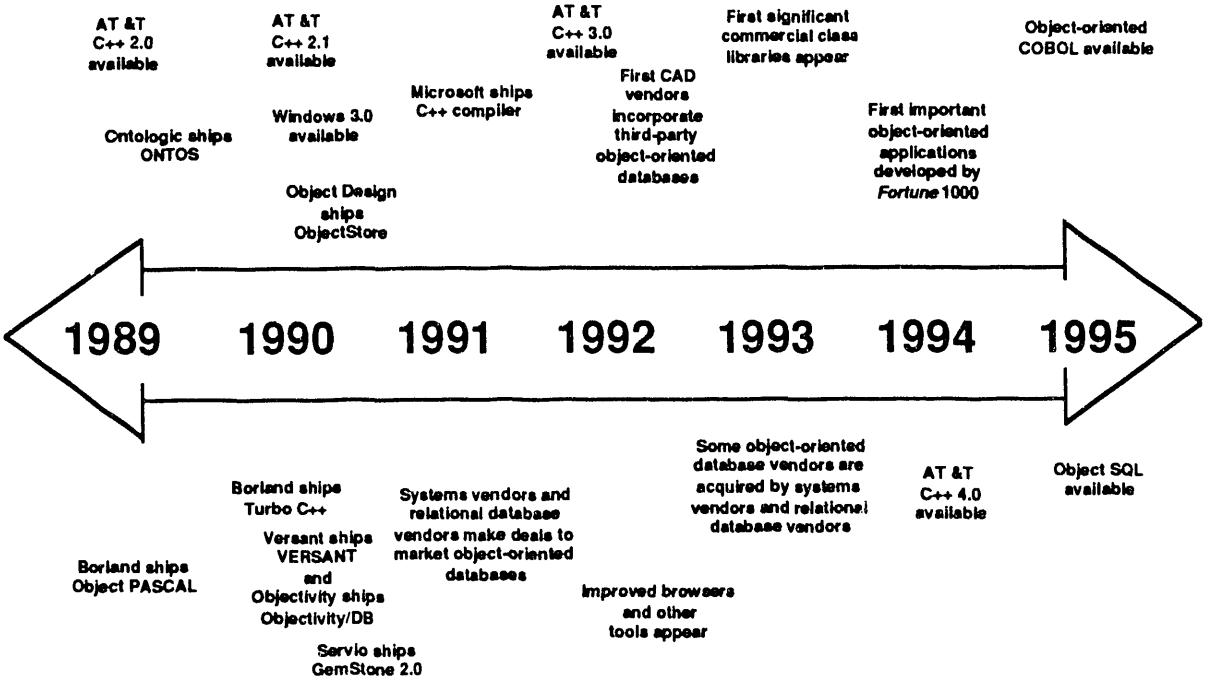


Figure 7-2. The Overall Development of the US Software Market

7.3. Problems to Overcome

There are a number of problems which must be overcome or allowed to evolve away before OOT can become part of mainstream software development. Some of the risks and costs of object-oriented application development were described in Section 4.6. Object technology is still maturing. Progress has been rapid, and the products are getting better in both capability and performance, but development of a new technology takes time. Figure 7-3 shows a timeline for

several years of intense activity in OOT. There are a number of vendors in the field who will not survive the inevitable shake-out, and so any organization must be cautious about making too much commitment to a single vendor's products, especially if it strays far from the developing consensus on standards.



Projected Object-Oriented Technology Timeline
taken from *Information Week*, January 7, 1991

Figure 7-3.

Another problem is the lack of tools to support object-oriented application development from analysis through maintenance. Powerful OOP environments have appeared (especially for Smalltalk), but more nearly complete coverage of the software life cycle is needed, especially for C++ development.

A very important unresolved issue is the best way to deal with legacy databases and applications. There have been a number of ideas on how to integrate OODBMS with the legacy databases, including the use of wrappers (see Figure 5-2).

Until there are standard versions of languages, databases, and development methodologies, widespread progress in adopting object technology will be limited. Currently, the vendors are still exploring all the possibilities and going in several directions with their products. It may take a few years for some areas to "settle down" to a consensus.

A whole different set of problems exists in our organizational environments, as was discussed in Section 4.7. Here, the issues are cultural. New paradigms for software development must be fostered with both training and management support. Software developers who have recently graduated from computer science programs will probably already be familiar with the new

technology, so we must make sure our older staff are not left behind, unable to work with advanced software development techniques.

An organization's reward structure must value reuse of other people's code and analysis/design segments, and skills of finding those items in a growing library of life cycle objects. This may be the hardest change to achieve.

New organizational structures may be required to build several levels of software components, and to accumulate libraries of generally useful components at each level. An organization can preserve and solidify its expertise in certain types of software (such as manufacturing or modeling or environmental management) by encoding expert design knowledge in discipline-related libraries and by storing these classes in a repository. We must also develop mechanisms for classifying and accessing those components when they are needed, to enable reuse.

Chapter 8:
Recommendations

8.1. Decision Whether to Change

This white paper has considered many aspects of object-oriented technology. What can be concluded from this survey? Is it time for us to be changing to object-oriented software development? Let us address four major decision areas raised by Ed Yourdon [YOUR90]:

1. *Is the object-oriented paradigm sufficiently mature, with well-developed products?*

   There is tremendous promise for improvement of software quality and development productivity, once the tools for software development mature so that the full life cycle is covered. Unfortunately, automated assistance in the analysis and design stages is still weak, though there are a growing number of books and courses on the subject. Object-oriented programming languages and tool sets are pretty well developed, with numerous products, books, courses, and such available. There are strong proponents and good reasons to go with either C++ or Smalltalk; the language decision must be made for each project individually. As application development environments compatible with our environment become increasingly available, we should take advantage of the productivity gains these tools offer. Object-oriented database products are still not mature, but already seem to be valuable for areas (like engineering) for which the relational approach is inadequate.

   The technology seems mature enough to use on problems for which conventional approaches are insufficient. Beyond that, for simple improvement of software quality and productivity, the state of the field is such that more progressive organizations would do well to adopt OOT as a recommended technology for projects of sufficient complexity. More conservative organizations may wish to do a couple of pilot projects to gain experience with the technology. As mentioned previously, security is an issue which is yet to be resolved.

2. *Is there good object-oriented implementation technology? Do we have tools available for effective use of the technology?*

   It is possible to do object-oriented analysis and design and then implement the system with conventional software tools, but it is not always straightforward. Many companies will choose to wait for significant use until there is an easy path from OO analysis and design to implementation in an OO language. It will not be very hard for C language programming shops to convert to C++, but it will be hard for Cobol shops to convert. (Yourdon says seventy-five percent of existing applications are written in Cobol, even though it is a 1950's era language.)

   However, help is on the way. Pure object languages are improving in performance and connectability to the conventional world, and some conventional languages are being extended to incorporate object features. Even Cobol has a standards committee planning changes to extend it into the object world; interim versions may be available in a couple of years.

3. *Is our computing organization sophisticated enough to change its methodology?*

   Watts Humphrey [HUMP89] has characterized five levels of "process maturity"[1] (see note below) which can be used to categorize any computing organization. A hallmark of level 3 is a documented, repeatable software development methodology. Parts of the DOE contractor organizations could be claimed to be almost at level 3, but it is a safe bet that level 3 process maturity is not consistent across all software development at any of the organizations. According to Yourdon and others, there is general agreement that an organization cannot make effective use of new tools or methodologies unless it is at level 3 or above. [YOUR90] One could probably argue with this judgement, however; even if an entire organization is not using a new technology most effectively, it seems likely that large, "process-mature" departments within an organization can use a disciplined and consistent approach and achieve much value from a new technology. However, we must also guard against falling back to level 2 because of adopting OOT methodology before it becomes standardized.

4. *Are our organization's systems and applications the kind that could effectively use OOT?*

   People and organizations generally don't change to a new technology just to solve familiar problems more quickly or efficiently. The cost of training, and changing the systems, and simple inertia generally encourage people to stay with the old technology until either the cost of not changing gets too high (and they could get order-of-magnitude improvements by changing), or new problems arise which cannot be feasibly solved with the old technology.

   What this means is that for batch systems or applications needing simple interaction via menus, or high transaction volumes with good performance or strict security requirements, it may be hard to justify the change to OOT. But for developing graphical user interfaces or a new application which cannot be solved easily with conventional techniques, OOT is a promising solution strategy.

Even if it is not time to change the whole organization's software to an object-oriented approach, it seems clear that the industry is evolving in that direction. A recent survey by International Data Corporation found that 70% of large U.S. corporations said they are programming with objects or plan to do so soon. The reason cited was money. Shearson Lehman Brothers claims a 30% drop in development costs and software that better simulates its business. Brooklyn Union Gas, weary of modifying and maintaining a huge and inflexible 13-year-old customer information system, scrapped it and created an object-oriented system that is 40% smaller and yet does more. With the flexibility inherent in OOP, they expect the new system to last twenty years at a fraction of the maintenance cost. [VERI91, p. 98]

---

[1] W. Humphrey's levels of process maturity:

1. **Initial level.** No formal methodology, no consistency, no standards.
2. **Repeatable level.** Intuitive; informal consensus, but no formalized methodology.
3. **Defined and managed level.** Formal, documented methodology; software inspections, configuration management.
4. **Measured level.** Level 3, plus formal process measurements used.
5. **Optimizing level.** Measurements from Level 4 used to improve process.

In the late 1980's, 85% of large U.S. computing organizations surveyed were in level 1, 10-12% at level 2, and about 3% at level 3. None were at level 4 or 5, as of 1990.

Given experiences such as this across the industry, we should encourage pilot projects to gain valuable experience with the technology. Some groups have already been doing this for some time (see Appendices for descriptions), and these groups are ready to build production systems wherever the tools and project conditions merit. With experience gained in such pilot projects, those computing professionals have learned which products are good for what uses, how to integrate object projects with the installed base of applications and data bases, and even some negative lessons of what doesn't work. We need to make effective use of the experience already gained, plus encourage other groups to do pilot project work with current versions of object tools and gain similar, updated experience.

8.2. Getting Started

A very important part of any effort to try object technology is education and training. First, people should be educated about the technology, its benefits and limitations, and its interaction with other technologies. All groups, from customers to management to developers, should be given appropriate education and training in the tools used. See Appendix X for a description of several courses used at the Savannah River Site.

A gradual, staged adoption of OOT can minimize the costs of wasted effort and the reduced productivity that accompanies the learning phase. During the coexistence of existing procedural languages and structurally developed systems, some authors [DUFF90] even recommend introducing object techniques by programming them within a traditional language. The argument is that this would produce useful code in a familiar language while helping the programmers to understand object-oriented programming language mechanisms, demystifying the technology, and increasing user acceptance.

There are problems with this approach, however. The pseudo-object system will be much harder to write and probably less efficient than if written in an object-oriented language. And the entire body of applications would still need to be reimplemented in object software, when the decision is made to adopt that technology. One of our experienced object developers warns that you should adopt the entire new object paradigm for the software life cycle; if you persist with part of the old software life cycle, you won't see the desired results.

Hybrid languages such as C++ are easier to change to, especially from C language, but programmers are more likely to continue old procedural programming habits in C++, yielding software modules that are hard to reuse and overall reduced benefits from the object technology. Additionally, C++ tools are not as mature as Smalltalk tools. However, applications in a hybrid language are easier to integrate with the "legacy" applications of an organization.

Pure object languages such as Smalltalk are cleaner and simpler to use, and are more likely to produce reusable code, since the object paradigm is the only one available. However, such systems are also much harder to integrate with legacy systems, since Smalltalk is a relatively "closed" language (that is, it cannot easily be called by or embedded within another system). In addition, some developers have found that Smalltalk development environments (from ParcPlace, at least) make it difficult to enhance a system's capabilities, for example the user interface, without significant extra effort.

8.3. Steps to Exploring the Technology

As object technology consultant Jon Hopkins says, OOT is a **revolutionary** technology, but adopting it must be an **evolutionary** process. Here are the necessary steps to start exploring the benefits of object-oriented technology.

Educate the organization on object technology and its potential impact on software development.

Events such as the Symposium on Object-Oriented Technology held in Oak Ridge in April 1991, which included excellent presentations by Jon Hopkins and Mary Loomis (see Appendix D for consultants listing), can help us in this regard. Regional professional societies such as ACM and IEEE Computer Society can also be helpful for technical education. An example of this is the Professional Development Seminar on Object-Oriented Technology which was held in Boulder in June 1991 [HERK91]; this seminar was conducted by two Martin Marietta employees.

Sell the object approach to all concerned:

- **senior managers** from whom you'll need money and support, and who can serve as champions in times of waning resources
- **middle managers** who must fit object-oriented projects in with the rest of the budget, schedule, and political realities
- **technical people**, who may not be uniformly receptive
- **customers (business people)**, whose understanding, support and good will is necessary for success.

Make sure that all these groups have **realistic expectations.**

Too often there is a tendency to skimp on training, but the new object-oriented paradigm requires a new mindset, so training and follow-up reinforcement are required. Managers will often be too busy to attend, so senior management must not only attend themselves, but must urge their subordinate managers to attend also.

For those organizations which have not already done so, use pilot projects to gain experience and to learn how best to adapt the object approach to local needs.

Find a bounded problem that cannot be handled easily with a conventional approach but which seems natural to consider with object techniques. It is best to choose a project which does not require close integration with a number of conventional systems. Consider future application possibilities, and give preference to a problem whose resulting class library might be reused on other problems.

Use consultants to assist in education, mentoring, and assisting with pilot projects.

This will bring people up to speed faster and reduce the amount of "wandering in the wilderness" which might otherwise occur. A rule-of-thumb agreed on by several OOT leaders at the 1991 OOPSLA conference was that 30% of an object-oriented project's people should already be experienced. In the early stages, this is hard to achieve without consultants. Excellent commercial consultants are available (see listing in Appendix D), but we should also consider internal mentoring programs to share the expertise of those who already have several years' experience in the area.

Work with some of your best people first, not the ones whom you can spare.

These people are your technical leading edge, and should be involved in your assessment of the technology. Even for excellent people, consultants indicate that it takes probably six months to change modes to object technology.

Make and follow a carefully thought out implementation plan.

List the objectives of the technology trial, for example to show: reduced cost, errors, redesign, maintenance; improved reuse, user friendliness, and so on. Keep track of project costs and benefits, differentiating such costs as training and consultation. Use early successes to promote support in later stages of the integration plan.

Keep up-to-date on the vendors and on standards efforts in the Object Management Group and ANSI.

The market is quite volatile. Products keep improving, so decisions based on product features and capabilities may change from month to month. Knowledge of progress toward standards will help to guard against committing to tools which are veering too far from the emerging standards.

Maintain good communication among all involved in working with object technology.

This is essential to maximize the usefulness of all the information being gained and the lessons learned. As the saying goes, "*None* of us is as smart as *all* of us." Information sharing will also reduce the time spent in working through technical problems and difficulties in interfacing between tools and systems. In Energy Systems, periodic "community meetings" and special interest newsletters in advanced technology areas such as knowledge-based systems have been used with good results. Also, user groups and technical newsletters are excellent ideas to promote learning about the new technology. In activities such as these, new developments in the marketplace can be discussed, and ongoing projects can be described and critiqued.

Begin building a software reuse library for the organization.

By 1993, each DOE site should set up a small group to: 1) establish naming conventions for the objects used to model the organization and 2) begin building the organization's reuse library. This activity will serve as a training ground for object-oriented technology, for the repository should be based on a commercial object database tool and will involve the use and evaluation of different OODBMS and object-oriented programming languages. Besides a small core group, additional people can be rotated through this activity both to augment the reuse staff and to aid technology transfer.

The reuse activity should mesh closely with any information engineering (IE) activity (especially the enterprise modeling) which is going on in the organization. We expect that the IE tools will evolve to greater use of OOT in the next couple of years, so the IE and reuse activities will merge when the IE/CASE tools allow it.

8.4. Cost of Achieving the Benefits

Much has been made of software reuse as a great benefit of OOT, but it does not come without significant investment. Software development practices and perhaps even the structure of software development organizations must change. As object technology is adopted, project teams must be allowed adequate **time** to study existing class libraries, in order to know what objects can be reused and how. This may appear to be "dead time" to observers, but it is essential to successful reuse.

It has been found best to give different groups on a project team different responsibilities with respect to objects. Some people will excel at building general purpose components, and some will be more skilled at reusing and customizing classes of objects for applications. The two groups should have significant interaction during development of an application, to ensure that developed classes are reused and not rewritten. Also, interaction between class builders and the application

developers will improve the probability that the classes in the class library will be at the right level of generality for the needs of the applications foreseen.

As experience with object-oriented projects grows, organizations should support reuse of the growing class libraries by forming groups for the purpose of building and maintaining general purpose classes, and managing class repositories as a corporate resource. This would be analogous to the Data Resource Administration groups found at various sites.

The accumulating libraries of reusable software modules will become a corporate resource of tremendous value which will increase our software productivity and enhance our ability to win contracts. The cost here is in setting up mechanisms to classify and later be able to retrieve the thousands of stored software modules and other "life cycle objects". These life cycle objects would include not just actual software, but also other useful items such as design segments.

One of the most widely recognized benefits of using object-oriented technology is the increased maintainability of the resulting software. The resources dedicated to maintenance will shrink significantly as the portion of object-oriented software in the installed base grows, and this is a permanent gain.

8.5. Changing the Organization to Object Technology

Some of the more revolutionary proponents of object-oriented software development argue that we should throw out structured software development completely, and get rid of the old installed base of legacy software as soon as possible. Some even suggest that the older structured software developers may be unsalvageable. However, the more moderate "synthesists" (Yourdon's term) argue that we can use the best ideas of object technology and structured technology at the same time. For example, we can identify a number of functions on local data stores in a data flow diagram and group those together as objects. However, the thinking process and communication between the user and developer is very different between the structured and object-oriented methods. There are indeed some fundamental concepts of software development which are just as relevant when using object technology as they are for structured development: notions like abstraction, partitioning and the conscious deferral of design decisions. Most of our experienced structured software developers will be able to shift to the new orientation.

And change will come, with or without our traditional computing organizations. Mainstream Information Age workers don't care about the lower level software constructs, or whether we are using object or structured techniques. What they need is the **results** of a different software development orientation-- higher level software components (robust and high quality, of course) from which they can assemble solutions to their information problems. Many of these workers will be able to use the emerging nontextual, nonprocedural visual software development tools which allow intuitive assembly of applications by manipulation of icons.

The world of software development will naturally reorganize into multiple layers, much as the hardware components market has (integrated circuits, chips, cards, etc.). The traditional computing organizations will provide the lower level components needed by information professionals, but additional levels of solution building may take place in either computing or end user organizations. This division of effort would do much to alleviate our perennial programmer shortage, because much needed software could then be assembled and used and passed on to others by less technical, more application domain oriented people. To support this new order, we need an object-oriented software architecture such that the modularity and interfacing mechanisms at each level fit the skills and interests of a distinct part of the reusable software components market.

There will still be some critical applications which must be developed by computing professionals, for example when high performance and tight integration with a production environment are required. But these applications too will benefit from increasingly robust and efficient object tools.

8.6. Conclusion

As object technology and its tools mature, so too will our understanding of how to manage a large and complex body of interrelated software components. Recalling an idea stated at the beginning of this paper, we see that reusable software components are corporate assets worthy of our best management skills. A repository of such software components, which is really an embodiment of an organization's collected expertise, will be an investment to sustain our future software efforts and help us meet the competition we face.

We should not wait for mature tools and standards before pushing the technology to more widespread use. We must learn as much as possible and be ready to exploit the competitive opportunities which will arise as new types of applications and appropriate tools emerge.

To do this, the authors strongly recommend that each DOE site organization embark on a software reuse program as described in Section 8.4. This should be done by 1993 with a minimum investment of 2-3 FTEs. The reuse group's efforts to create an object repository for the organization should be coordinated closely with any enterprise modeling efforts already under way using information engineering methodology. We feel that we cannot be ready to make timely and effective use of object technology if we do not make this technology investment soon.

References

[AHME91] Ahmed, Shamim, Albert Wong, Duvvuru Sriram and Robert Logcher, A Comparison of Object-Oriented Database Management Systems for Engineering Applications, Massachusetts Institute of Technology Research Report R91-12, May 1991.

[ARNO91] Arnold, Patrick, Stephanie Bodoff, Derek Coleman, Helena Gilchrist, and Fiona Hayes, "An Evaluation of Five Object-Oriented Development Methods", Hewlett Packard Laboratories Technical Report HPL-91-52, June, 1991.

[ATKI89] Atkinson, Malcolm, Francois Banchilon, David DeWitt, Klaus Dittrich, David Maier, and Stanley Zdonick, "Object-Oriented Database System Manifesto," *Proc. of the First International Conference on Deductive and Object-Oriented Databases*, Elsevier, Berlin, 1989.

[ATWO89] Atwood, Tom, OODBMS Column in *Hotline on Object-Oriented Technology*, several issues in 1989-1990

[BLAK90] Blakeley, Jose', Craig Thompson and Abdallah Alasqhur, OQL(X): Extending a Programming Language X with a Query Capability, Texas Instruments Information Technologies Laboratory, Technical Report 90-0701, November 20, 1990.

[BOOC91] Booch, Grady, *Object-Oriented Design with Applications*, Benjamin/Cummings, Redwood City, CA, 1991.

[COAD91a] Coad, Peter and Edward Yourdon, *Object Oriented Analysis*, 2nd ed., Yourdon Press, Englewood Cliffs, NJ, 1991.

[COAD91b] Coad, Peter and Edward Yourdon, *Object-Oriented Design*, Yourdon Press, Englewood Cliffs, NJ, 1991.

[COX 90a] Cox, Brad, "There is a Silver Bullet," *Byte*, October 1990.

[COX 90b] Cox, Brad, "Planning the Software Industrial Revolution," *IEEE Software*, November 1990.

[COX 91] Cox, Brad, *Object-Oriented Programming*, 2nd ed., Addison-Wesley, Reading, Mass., 1991.

[CUNN86] Cunningham, Ward and Kurt Beck, "A Diagram for Object-Oriented Programs," In *Proc. of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Portland, OR, 1986.

[DABR90] Dabrowski, Christopher, Elizabeth Fong and Deyuan Yang, "Object Database Management Systems: Concepts and Features", National Institute of Standards and Technology, Special Publication 500-179, 1990

[DECH91] de Champeaux, Dennis, "A Comparative Study of Object-Oriented Analysis Methods," Hewlett Packard Laboratories Technical Report HPL-91-41, April 1991.

[DEMA78] De Marco, Tom, *Structured Analysis and System Specification*, Yourdon, Inc., New York, 1978.

[DUFF90] Duff, Chuck and Bob Howard, "Migration Patterns," *Byte*, October 1990.

[GIBS90] Gibson, Elizabeth, "Objects Born and Bred," *Byte*, October 1990.

[HARM90a] Harmon, Paul, "A Brief Overview of Software Methodologies," *Intelligent Systems Strategies*, Vol. VI, No. 1, January 1990.

[HARM90b] Harmon, Paul, "Object Oriented Systems," *Intelligent Systems Strategies*, Vol. VI, No. 9, September 1990.

[HARM91a] Harmon, Paul, "A Brief Overview of Software Methodologies," *Intelligent Systems Strategies*, Vol. VII, No. 1, January 1991.

[HARM91b] Harmon, Paul, "The Year in Review: The Market for Intelligent Software," *Intelligent Systems Strategies*, Vol. VII, No. 2, February 1991.

[HARM91c] Harmon, Paul, "CASE and the Future of Expert-Systems Building Tools," *Intelligent Systems Strategies*, Vol. VII, No. 4, April 1991.

[HARM91d] Harmon, Paul, "What's Happening in Computing," *Intelligent Systems Strategies*, Vol. VII, No. 8, August 1991.

[HARM91e] Harmon, Paul, "The Object-Oriented Market in the Fall of 1991", *Object-Oriented Strategies*, Premier Issue, October 1991.

[HUMP89] Humphrey, Watts, *Managing the Software Process*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[LYNG91] Lyngbaek, Peter et. al., "OSQL," Hewlett-Packard Laboratories, Technical Report HPL-DTD-91-4, January 15, 1991.

[KIM 89] Kim, Won and Frederick Lochovsky (eds.), *Object-Oriented Concepts, Databases, and Applications*, ACM Press, New York, 1989.

[MARQ91] Marquess, Philip, "Paradigm," *DEC Professional*, March 1991.

[MEYE88] Meyer, Bertrand, *Object-Oriented Software Construction*, Prentice-Hall, Englewood Cliffs, NJ 1988.

[MOOD91] Moody, Scott, "Library Clashes", *Object Magazine*, vol. 1 no. 1, May/June 1991, pp. 13-18.

[MOOR91] Moore, Ken, Craig Thompson, and William Kent, eds. "X3/SPARC/DBSSG/OODBTG Recommendations for Standards in Object Information Management", Revision 7, Accredited Standards Committee X3, INFORMATION PROCESSING SYSTEMS, Document No. OODB 90-R7.

[PRIE91] Prieto-Diaz, Ruben, "Making Software Reuse Work: An Implementation Model", ACM SIGSOFT *Software Engineering Notes*, vol. 16 no. 3, July 1991, p. 61-68.

[RUMB91] Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy and William Lorenson, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[SHLA88] Shlaer, Sally and Stephen Mellor, *Object-Oriented Systems Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[STEF86] Stefik, Mark, Daniel Bobrow and Kenneth Kahn, "Integrating Access-oriented Programming into a Multi-paradigm Environment," *IEEE Software*, 1986.

[THOMA91] Thomas, David A., "Object Utopia: a View of Object-Oriented Computation in the 21st Century," *Object Magazine*, November/December 1991, pp. 10-18.

[THOMP91] Thompson, Craig, "Object Data Model Reference Model", Revision R7,Accredited Standards Committee X3, INFORMATION PROCESSING SYSTEMS, Document No. OODB 89-01R7.

[VERI91] Verity, John and Evan Schwartz, "Software Made Simple," *Business Week*, September 30, 1991, pp. 92-100.

[WIRF89] Wirfs-Brock, Rebecca and Brian Wilkerson, "Object-Oriented Design: A Responsibility-Driven Approach," In *Proc. of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*, New Orleans, LA, 1989.

[YOUR77] Yourdon, Edward and Larry Constantine, *Structured Design*, 2nd ed., Yourdon, Inc., 1977.

[YOUR90] Yourdon, Edward, "Auld Lang Syne," *Byte*, October 1990.

[ZDON90] Zdonick, Stanley and David Maier, *Readings in Object-Oriented Database Management Systems*, Morgan Kaufmann Publishers, San Mateo, CA, 1990.

Appendix A:

Glossary[1]

Some of the terms used in this report are found throughout the literature, occasionally with conflicting meanings. This Appendix contains short, informal definitions of key concepts and terms.

Abstract Data Type:

A programming technique that defines a data space, hiding procedures and details the programmer does not need to know to manipulate the data. The definition of an abstract data type consists of an internal representation along with a set of procedures required to access and manipulate the data.

Active Database:

A database system in which retrieval and update operations result in invocation of procedures. Such procedures, known as triggers, are associated with particular fields. When the field is accessed, the trigger is activated.

Attribute:

Attributes are properties of an entity. An entity is said to be described by its attributes. In a database, the attributes of an entity have their analogues in the fields of a record. In an object database, instance variables may be considered attributes of objects.

Class:

A generic description of an object type consisting of instance variables and method definitions. Class definitions are templates from which individual objects can be created.

Class Object:

A class definition. In many OOPL and ODBMS implementations, class definitions are objects that are instances of a generic class, or metaclass.

Class Hierarchy:

Classes can naturally be organized into structures (tree or network) called class hierarchies. In a hierarchy, a class may have zero or more superclasses above it in the hierarchy. A class may have zero or more classes below, referred to as its subclasses.

---

[1]Source: C. E.Dabrowski et al., "Object Database Management Systems: Concepts and Features," *National Institute of Standards and Technology Special Publication 500-179*, April 1990; reprinted with permission from Elizabeth N. Fong, National Institute of Standards and Technology.

Class Library:

A set of related classes belonging to a specific domain. For example, a graphics library may exist, consisting of classes of graphical objects.

Code Reuse:

The ability to use a single piece of code for more than one purpose in a computer application. When a superclass definition is inherited by a subclass, the code associated with the superclass definition, including method definitions, is reused in the subclass. Code reuse has the effect of reducing the amount of code needed to implement an application.

Composite or Complex Object:

An object which is made up of other objects. Composite objects consist of collections of parts, each of which is itself an object. Each part is in an "Is-Part-Of" relationship with the object of which it is a component.

Concurrency Control:

A mechanism that regulates access to objects and prevents users from executing inconsistent actions on the database.

Data Abstraction:

A programming technique by which the internal representation and operations of an object are made only partially visible, allowing only certain information relevant to a particular application to be seen. The actual methods by which computations are performed remain hidden from external view. See also encapsulation.

Data Model:

The data model is a specification of the structure of the database, the operations and the integrity rules.

Database Schema:

The complete set of individual schema definitions which describe the logical structure of a database. In an ODB, the database schema is expressed in the set of class definitions for a database.

Dynamic Binding:

Also known as run time binding or late binding. Dynamic binding refers to the association of a message with a method during run time, as opposed to compile time. Dynamic binding means that a message can be sent to an object without prior knowledge of the object's class.

Encapsulation:

The packaging of data and procedures into a single programmatic structure. In object-oriented programming languages, encapsulation means that an object's data structures are hidden from outside sources and are accessible only through the object's protocol.

Entity:

A collection of information items which can conceptually be grouped together and distinguished from their surroundings. An entity is described by its attributes. Entities can be linked, or have relationships to other entities.

Extensible Database Management Systems:

A class of DBMS incorporating additional data modeling capabilities together with data management services needed for application domains which cannot easily make use of conventional DBMS.

Extensibility:

The ability to dynamically augment the database schema. This includes addition of new data types and class definitions for representation and manipulation of unconventional data such as voice data, image data, and data associated with artificial intelligence applications.

Generalization:

Refers to the relationship between a superclass and its subclasses. A superclass is a generalization of its subclasses.

Handle:

A pointer to, or address of, an object. A handle is a unique, and nonchangeable reference to an object. In some systems, the term handle is interchangeable with the term object identity .

Inheritance:

A mechanism which allows objects of a class to acquire part of their definition from another class (called a superclass). Inheritance can be regarded as a method for sharing a behavioral description.

Instance:

An individual occurrence of an object.

Instance Variable:

An attribute of an object. A class definition may specify the set of instance variables that constitute the data structures for objects of the class.

Message:

See message passing.

Message Passing:

The means by which objects communicate. Individual messages may consist of the name of the message, the name of the target object to which its being sent, and arguments, if any. When an object receives a message, a method is invoked which performs an operation that exhibits some part of the object's behavior.

Method:

A method is the body of code executed in response to a message. The methods associated with a class definition effectively describe the behavior of all the instances of the class.

Multiple Inheritance:

The ability for a class to inherit from more than one superclass. Thus, a class may inherit instance variables and methods from multiple superclasses.

Object:

An object is the basic unit of computation. An object has a set of "operations" and a "state" that remembers the effect of operations. Classes define object types. Typically, objects are defined to represent the behavioral and structural aspects of real world entities.

Object ID (object identity):

A permanent unique identifier that is assigned to each object. The identifier is independent of the value of the instance variables of the object, and remains constant despite any change in the object's state. Object Identity is sometimes used interchangeably with the term handle.

Object Server:

An Object Server is the software system which supports transaction management and storage management functions for objects.

Part Hierarchy:

A hierarchy of component objects which form parts of a composite object. A composite object will be made up of objects which may themselves have components. This is distinguished from a class hierarchy which consists of classes related through inheritance.

Persistence:

A property of data or objects implying that it has a lifetime greater than the process which created it.

Persistent Object Store:

The object database, or ODB.

Polymorphism:

Polymorphism refers to being able to apply a generic operation to data of different types. For each type, a different piece of code is defined to execute the operation. In the context of object systems, polymorphism means that an object's response to a message is determined by the class it belongs to.

Protocol:

The set of messages an object will respond to. The term protocol can sometimes be used interchangeably with the term public interface. In an ODBMS, protocols are specified in class definitions.

Referential Integrity:

In a relational database, referential integrity means that no record may contain a reference to the primary key of a nonexisting record.

Run Time Binding:

See Dynamic Binding.

Shadowing:

The definition of a method in a class description to replace a method that would otherwise be inherited from a superclass. When a message is sent to an object that is an instance of the subclass, the method defined in the subclass is invoked. The shadowed method in the superclass is not invoked.

Specialization:

Refers to the relationship between a subclass and its superclasses. A subclass is a specialization of its superclasses.

State:

The set of values for the instance variables of an object. When the values of any of the object's instance variables change, the object's state is altered.

Subclass:

When a class inherits the instance variables and methods from another class, it is referred to as its subclass.

Superclass:

The class from which the instance variables and methods of a subclass are inherited.

Appendix B:

Object-Oriented Languages

Several of the following language descriptions of object-oriented languages have been taken from [MARQ91]:

Ada

Ada is actually an object-based language. Although Ada supports data abstraction and information hiding, it does not provide inheritance or dynamic binding. Typically, class definitions are placed in a package specification.

There are toolsets which are an extension of the Ada language and provide many of the object-oriented capabilities of languages such as C++ and Smalltalk. One such product is Classic-Ada from Software Productivity Solutions of Indiatlantic, Florida. Classic-Ada translates object-oriented constructs such as class definitions and polymorphic functions into Ada source code which can then be compiled. Other toolsets offered by the vendor support persistent objects, interactive browsing and debugging, and reusable class libraries.

Simula

Simula 67 was the first object-oriented language. A hybrid language based on Algol 60, it introduced encapsulation and inheritance. It supports coroutines, in which an object can perform actions independently of other objects. Popular in the Scandinavian countries, a number of compilers are commercially available.

Smalltalk

Originated at the University of Utah and later at Xerox PARC in the early 1970s, Smalltalk evolved into its present form at Xerox PARC later in the decade. It combines many of the ideas from Simula with the typeless style of LISP. It is a pure language which even represents integers as objects. In Smalltalk, classes themselves are objects which are instances of a higher level class, i.e., a metaclass.

C++

This language is a hybrid extension of C developed at AT&T Bell Laboratories. Dynamic binding is available only when requested. When defining a class, those methods which are subject to redefinition by descendant classes are specified as virtual. For all other routines, the compiler generates static calls. There is no automatic garbage collection in C++. Typically, this is performed through the definition of constructor and destructor functions for each class of objects. C++ is rapidly becoming the de facto standard of object-oriented languages due to the proliferation of the C language; however, development in C++ may be more difficult than in other languages such as Smalltalk.

CLOS

The Common LISP Object System (CLOS) is a hybrid extension of the Common LISP standard adopted by the ANSI X3J13 committee. CLOS introduces generic functions as a means to invoke methods associated with an object. By using generic functions, CLOS can write methods that specialize multiple parameters. These multi-methods are intended for those operations whose implementation depends upon the type of more than one argument. Auxiliary methods (i.e., before, after, around) can be used to extend existing

methods by attaching supplemental procedures to them rather than modifying them. The metaobject protocol allows software developers to provide new of different behaviors of classes, e.g., methods of object creation, inheritance strategies. CLOS is a powerful prototyping language favored in the artificial intelligence community, but its popularity does not extend into the commercial sector.

Eiffel

Eiffel is a pure object-oriented language which embodies many of the ideas in Bertrand Meyer's Object-Oriented Software Construction. Eiffel supports multiple inheritance and strong type checking, and parameterized types, i.e., referred to as genericity in Eiffel. It includes Boolean assertions that can be evaluated at run-time. Assertions can be used to verify that an object's properties. Using assertions, pre- and post-conditions can be associated with a method. The former gives those events that must be satisfied before calling the routine; the latter dictates certain events that must occur after the routine has completed. Eiffel is a limited following is the United States; however, ESPRIT has recently announced a major research initiative to generate a library of class objects for MIS-related activities in the Eiffel language. This should promote interest in the Eiffel language both here and abroad.

Appendix C:

Recommended Readings

Object-Oriented Analysis

Coad, Peter and Edward Yourdon, *Object-Oriented Analysis*, 2nd ed., Yourdon Press, Englewood Cliffs, NJ, 1991.

Cunningham, Ward and Kurt Beck, "A Diagram for Object-Oriented Programs," In *Proc. of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Portland, OR, 1986.

Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy and William Lorenson, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

> Provides an overview of the Object Modeling Technique (OMT). Offers several detailed examples of their analysis technique. While their data modeling is similar to that espoused by [Shlaer and Mellor, 1988], the authors provide more thorough coverage of providing the behavior that an application must exhibit.

Shlaer, Sally and Stephen Mellor, *Object-Oriented Systems Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1988.

> Relies heavily on the relational data model. It concentrates on data modeling and neglects the modeling of behavior. Several reviewers have suggested that this method more closely resembles semantic data modeling than object-oriented analysis.

Object-Oriented Design

Booch, Grady, *Object-Oriented Design with Applications*, Benjamin/Cummings, Redwood City, CA, 1991.

Coad, Peter and Edward Yourdon, *Object-Oriented Design*, Yourdon Press, Englewood Cliffs, NJ, 1991. [The companion to *Object-Oriented Analysis*.]

Cox, Brad, *Object-Oriented Programming*, 2nd ed., Addison-Wesley, Reading, Mass., 1991.

Meyer, Bertrand, *Object-Oriented Software Construction*, Prentice-Hall, Englewood Cliffs, NJ 1988.

> The first few chapters provide the motivation for adopting the object-oriented paradigm for software. Remaining chapters discuss the Eiffel programming language.

Wirfs-Brock, Rebecca and Brian Wilkerson, "Object-Oriented Design: A Responsibility-Driven Approach," In *Proc. of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*, New Orleans, LA, 1989.

Wirfs-Brock, Rebecca, Brian Wilkerson, and Lauren Wiener, *Designing Object-Oriented Software*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

Recommended.

Object-Oriented Programming

Cox, Brad, *Object-Oriented Programming*, 2nd ed., Addison-Wesley, Reading, Mass., 1991.

Ellis, Margaret and Bjarne Stroustrup, *The Annotated C++ Reference Manual*, Addison-Wesley, Reading, Mass. 1990.

The ANSI Base Document for the C++ programming language. Not for beginners.

Goldberg, Adele and David Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading, Mass. 1983.

Keene, Sonya, *Object-Oriented Programming in Common LISP*, Addison-Wesley, Reading, Mass., 1989.

An excellent text on object-oriented programming in the Common LISP Object System (CLOS) even though CLOS specifications have changed since its publication. There is little mention of the metaobject protocol since the text predates its specification.

Lippman, Stanley, *C++ Primer*, Addison-Wesley, Reading, Mass., 1989.

Weiner, Richard and Lewis Pinson, *An Introduction to Object-Oriented Programming and C++*, Addison-Wesley, Reading, Mass., 1988.

Object-Oriented Database Management

Kim, Won and Frederick Lovchovsky eds., *Object-Oriented Concepts, Databases, and Applications*, ACM Press, Reading, Mass., 1989.

A collection of papers on various aspects of object technology. Slightly dated, this publication is geared to academia.

Parsaye, Kamran, Mark Chignell, Setrag Khoshafian, and Harry Wong, *Intelligent Databases: Object-Oriented, Deductive Hypermedia Technologies*, Wiley & Sons, New York, 1989.

Zdonick, Stanley and David Maier eds., *Readings in Object-Oriented Database Management Systems*, Morgan Kaufmann, 1989.

A collection of papers on various aspects of object data management. An excellent reference text, but not suitable for beginners.

Recommended Periodicals

International OOP Directory, SIGS Publications, phone 212:274-0640.

> Detailed encyclopedia with reprints of significant articles, company briefs, comprehensive product listing, service directory, and extensive bibliography. Excellent single-source information reference. Second edition due soon.

Journal of Object-Oriented Programming, ISSN #0896-8438, published bimonthly by SIGS Publications, Inc., 310 Madison Avenue, Suite 503, New York, New York 10017, phone 212:972-7055.

Object-Oriented Strategies, focused on O-O products, from languages and tools to operating systems and databases (newly split off from *ISS*, below)
Intelligent Software Strategies, focused on expert systems intelligent CASE, natural language, neural networks

> An excellent source of product reviews and comparisons and state-of-the-art surveys on all the topics covered. Both above newsletters from (Paul) Harmon Associates, about $370 per year. Circulation info: published by Cutter Information Corp., phone 617:648-8700.

Object Magazine, focused on technical, business, and organizational issues involved in adopting object technology. $25 per year for individuals, $59 for institutions. Subscriber Services, Dept. OBJ, P.O. Box 3000, Denville, NJ 07834-9970.

Recommended Conferences

OOPSLA - Object-Oriented Programming Systems, Languages, and Applications
> More academic oriented than the others.

Object World
> For the commercial world.

SCOOP - Seminars & Conference on Object-Oriented Programing. Both West (spring) and East (late summer/fall) versions. Commercial world oriented.

Appendix D:

Consultants

This is a partial list of consultants on object technologies, with a few annotations. In addition to these, many of the tool and database vendors have excellent technical staff, many of whom are willing to give short presentations on object technologies and their respective products.

Knowledge Systems Corporation
114 MacKenan Drive, Suite 100
Cary, North Carolina 27511-6446
phone 919:481-4000

Reed Phillips, Sam Adams, Ken Auer. Smalltalk.

Palladio Software
Suite 360
16535 W. Blue Mound Road
Brookfield, Wisconsin 53005
phone 414:789-5253

Jon Hopkins, frequent speaker at object conferences. Presented seminar at Energy Systems Symposium, April 1991.

Technology Exchange Company
Route 128
One Jacob Way
Reading, Massachusetts 01867
phone 617:944-3700

C++ training. Beta version of their OOP & C++ course offered at Energy Systems in 1990.

The Object People, Inc.
91 Second Avenue
Ottawa, Ontario
Canada K1S 2H4

Smalltalk

Berard Software Engineering, Inc.
18620 Matenay Road
Germantown, Maryland 20874
phone 301:417-9884

Object-oriented software engineering

Semaphore Training
800 Turnpike Street
North Andover, Massachusetts 01845

phone 508:794-3366

C++

Instantiations, Inc.
921 SW Washington
Suite 312
Portland, Oregon 97205

phone 503:242-0725

C++ and Smalltalk

Elite Systems
7400 SW Barnes Road #911
Portland, Oregon 97225

OO design and C++

Quality Software Engineering
P. O. Box 303
Beaverton, Oregon 97075-0303

phone 503:645-6698

OO design and C++

Pantheon Systems, Inc.
P. O. Box 230835
Portland, Oregon 97223

phone 505:626-8639

Smalltalk

Rational Consulting
3320 Scott Boulevard
Santa Clara, California 95054-3197

phone 303:986-2405

Grady Booch. C++ and Ada.

Robert Fulton
Professor of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

phone 404:894-7409

CALS, IDEF

Appendix E:

Object-Oriented Products[2]

The following table is an extensive listing of object-oriented products available as of late 1991. The table is taken from the premier issue (October 1991) of the *Object-Oriented Strategies* newsletter, courtesy of Editor Paul Harmon and Associate Editor Curtis Hall, who granted us permission to use it. For more information about this newsletter, see Appendix C.

OBJECT-ORIENTED PRODUCTS BEING SOLD IN THE US

In the course of each year, we will try to review all the OO products on the market and provide information so that readers can keep up with the variety and features of the products in each niche.

In this issue, we include tables on four groups of products: (1) OO Application Development Environments and Tools, (2) OO Interface Development Tools, (3) OO Design Tools, and (4) OO Databases.

Table 1 — Object-Oriented Application Development Environments and Tools

Name of Product (Vendor)	Price	Hardware (Operating System)	Comments
Application Manager **Smalltalk/286** **Smalltalk/VPM and V/Win** **Smalltalk/V-Mac** (Coopers & Lybrand Softpert Systems Div.)	$395 $475 $395	IBM PC and compatibles, Macintosh.	For developing Smalltalk/V applications. Includes functions to allow developers to view and access the subset of classes and methods that are considered part of an application package; provides software engineering support for application development. Source control and change browsers also available.
ENFIN/2 (Enfin Software Corp.)	$995- $4,900	Three environments supported: OS/2/Presentation Manager, DOS/Windows, and Unix/Motif.	OO development environment (based on an enhanced Smalltalk paradigm) containing class browser, inspector, interactive debugger; includes: set of reusable classes and methods; 4GL development tools such as interactive interface builder, graphical report generator, business graphic objects, an SQL query editor, and a DB and table definition facility. A program generator is also available as an add-on package.
LabVIEW 2 (National Instruments)	$1,995	Mac with 4 mb RAM; 80 mb hard drive recommended.	Graphic programming language for developing instrumentation applications. Features hierarchical menus, wirestretching, and complete clipboard capabilities. Includes graphical compiler for generating optimized machine code from block diagrams, ability to link-in external code routines.
Object Designer (Chen & Associates)	$495	IBM PC/AT and compatibles; requires 640K RAM and EGA/VGA graphics.	Available in three language versions: Turbo Pascal 5.5; C++, and C++ and Turbo Pascal 5.5. Features: object hierarchies, object variable definitions, and object method specifications; integrated data dictionary, continuous error checking, and code generation.

[2]Source: Paul Harmon, ed., *Object-Oriented Strategies*, Premier Issue, October 1991. Used with permission by Associate Editor Curtis Hall.

Table 1, cont. — Object-Oriented Application Development Environments and Tools

Name of Product (Vendor)	Price	Hardware (Operating System)	Comments
ObjectCraft (V. 2.0) (ObjectCraft, Inc.)	$399	IBM PC/AT or compatibles (DOS); VGA/EGA graphics.	Lets users develop OO programs visually by "drawing" a program's objects, flow, interface, and DB connections on screen. These diagrams can be automatically converted to compilable C++ or Turbo Pascal 5.5 code. Features include: predefined interface objects (buttons, switches, etc.), capabilities for library development, and the ability to import existing C++ files and write C++ methods inside the ObjectCraft environment.
ObjectPlus (ProtoSoft Inc.)	$995-$4,500	IBM PC/AT or compatibles (DOS); requires MS Windows.	OO analysis and design tool and code generator operating in Windows; available for Ada, C, C++, and Object Pascal code generation. Features include full cycle support, and DB integration with Oracle, dB2, etc.
Objectworks Smalltalk **Objectworks C++** (ParcPlace Systems)	$3,500 $3,000	386 PCs (DOS), (Windows), Apollo, Sun, HP 9000 series 300 (Unix), DECstation, Mac, Sun SPARCstation. C++ version available only for Sun 3 & 4 and SPARCstations.	Programming language/toolset. Supports portability of source code, objects, and binary code. Features: wide range of application portability, reusable class library, complete source code to class library. C++ version supports creation of abstract/concrete classes, traditional C data structures and procedures, type checking, multiple inheritance, dynamic memory allocation, data abstraction.
Prograph (v. 2.5) (TGS Systems)	$495	Any Mac with 2 mb RAM; hard disk; system 7.0, 6.0, A/UX.	OOP environment that allows developer to "draw" application. Includes: graphical dataflow language and built-in source code debugger for building "clickable" applications, interpreter/compiler, WYSIWYG editor for associated methods, customizable interface elements, full single-inheritance support. Also includes DB engine for creating multiuser flatfile, relational, or object DB applications.
Object-Oriented Structured Design/C++ Notation Editor (Interactive Development Environments)	Varies by platform	DecVAX (VMS), Sun, and Apollo (Unix) workstations.	OOSD/C++ Notation Editor supports the OOSD notation for designing C++ applications. Features specific to C++ have been added. The editor provides full support to classes, single and multiple inheritance, member functions, templates, global and local scoping, etc. Future enhancements will connect designs created with this editor to the Software Through Pictures (IDE) multi-user repository for functions including quality assurance, code and design generation, and reuse of design and code.
SPOKE (Expertelligence Inc.)	$13,000	Sun SPARCstation.	OOP environment with multiple inheritance, dynamic classes, generic functions. Written in C and generates C code. May be used in conjunction with Expertelligence's Action! interface builder.
Teamwork (Cadre Technologies, Inc.)	Varies widely	Available on a wide variety of operating platforms.	Modeling and design tool for several application development areas: information modeling — helps systems analysts and DB designers model the entities, relationships, and attributes of all application data at the conceptual level; systems analysis — for real-time structured analysis; and systems design — for employing structured design techniques.

Table 2 — Object-Oriented Interface Development Tools

Name of Product (Vendor)	Price	Hardware (Operating System)	Comments
Action! (Expertelligence)	$1,995-$35,000	MicroExplorer, Macintosh, Sun SPARCstation (Unix), IBM PS/2 (Presentation Manager).	Based on ExpertObject Code (a Lisp code containing the Flavors definition of each of the 13 objects found in the standard Macintosh interface, windows, menus, keyboard events, etc.). Allows "point-and click" development of GUIs which include connections between interface objects and underlying functional code. Features: inspectors, browsers, and dynamic class editors; supports multiple inheritance, exception handling, multi-method dipatch, and modularity. Applications developed on one platform may be ported to others without modification.
AppMaker (Bowers Development Corp.)	$295	Mac Plus or larger running System 5.0 or later, also available for A/UX (MacUnix) and Apple IIGS. Language system (MPW or THINK, etc.) required for compiling and linking of generated code.	Tool for creating and changing Macintosh applications quickly with a minimum of programming. Provides point-and-click user interface development, including generation of complete, "ready-to-run" source code in Pascal or C (for entire user interface or selective modules). Features: GUI editor for arranging elements on screen; code generator can be customized to support individual program, methods to support cross-platform development, or to add languages; resource editor lets you open and enhance existing applications as well as create new ones; includes library of over 70 routines in source code form.
CommonView 2 (ImageSoft Inc.)	$399-$599	IBM PCs and compatibles.	Application framework of existing and reusable C++ classes for developing GUI-based applications. Allows developers to create a single, portable GUI that can run under Windows, Presentation Manager, HP New Wave, OSF/Motif, and Mac. Pre-tested objects may be reused to create customized (application-specific) objects.
C_talk/Views (CNS Inc.)	$450	IBM PCs or compatibles: requires 512 K RAM. Requires MS C compiler, MS Windows 2.1 or later, MS Development Kit. Works with most C compilers.	For MS Windows program development in C. The most common elements of Windows programs are packaged as reusable software components. Allows Windows functions to be converted into standard C code.
C++/Views (CNS Inc.)	$495	IBM 286, 386, or compatibles; requires MS Windows 3.0 and SDK, C++ compiler/reprocessor 2.0.	For developing Windows 3.0 applications in C++. Features: 65 C++ classes encapsulating Windows functionality, class hierarchy browser supporting single and multiple inheritance, source/file editing, multiple rooted hierarchies, adding/deleting classes, application/library management, automatic make file, and class generation and documentation.
MacApp (v. 2.0) (Apple Computer Inc.)	$275	Mac Plus, SE or II with 2 mb RAM and hard drive. Requires Mac Programmer's Workshop (MPW) 3.0 and MPW Pascal 3.1, MPW C 3.1 and MPW C++ 3.1 or Think Pascal 3.0.	OO toolkit consisting of an object library from which users may modify existing objects for creating Macintosh applications, including scrollable, resizable windows, and multi-page printing. Applications "inherit" the look and feel of standard Mac-based applications directly from the MacApp code. Includes tools for source code browsing and creating and editing windows and dialog boxes. Operates on all Mac hardware and runs under MultiFinder and the A/UX operating system.

Table 2, cont. — Object-Oriented Interface Development Tools

Name of Product (Vendor)	Price	Hardware (Operating System)	Comments
MacDialog Editor (Quintus Corp.)	$325	Mac +, SE, or II; MacOS 6.0, MacProlog 6.0 or later. Hard disk recommended.	Mouse-driven development tool for creating interfaces for Mac-based Prolog applications, including: radios, buttons, checkboxes, menus, text fields, edit fields, icons, picture buttons, and picture checkboxes. Supports multiple scrollable menus.
NeXtStep (NeXt Inc.)	NA	NeXt machine, IBM PS/2, and RS 6000.	Window-based development environment consisting of four components: Window Server and Workspace Manager — for file and system navigation; Application Kit — a set of approx. 25 software classes that define interactive objects such as windows, buttons, and scrollers; and Interface Builder — uses previously created software objects to develop user interfaces. Uses Display PostScript system for both screen display and printing.
ObjectVision (Borland)	$99.95	IBM PCs and compatibles.	Allows developers to create MS Windows applications visually by combining declarative style logic (decision trees and intelligent prompting) with a form-oriented interface, visual programming techniques, incremental development method, and interpreter with graphical toolset.
Object/1 (Micro Data Base Systems Inc.)	$995	IBM AT or 386; OS/2 and Presentation Manager or DOS/Windows 3.0; 4 mb RAM, EGA/VGA, mouse.	For developing graphical interfaces for OS/2 Presentation Manager and MS Windows 3.0 applications. Includes OOP language rooted in C, nearly 300 classes, and 3,000 methods. Applications may operate stand-alone using TBL DB engine (KnowledgeMan and GURU DB), or the client side for MDBS IV, MSoft/Sybase, or ORACLE.
Turbo Vision Development Toolkit (Blaise Computing, Inc.)	$149	IBM PCs and compatibles.	Set of utilities and object libraries for use with the Turbo Vision (Turbo Pascal 6.0) OO, event-driven framework. Includes resource editor for interactively creating and modifying dialog boxes and other resources, a utility to convert Turbo Vision resources into Windows resource script files, and an object library that extends Turbo Vision's capabilities.

Table 3 — Object-Oriented Design Tools

Name of Product (Vendor)	Price	Hardware (Operating System)	Comments
OOATool Commercial Version **OOATool Small Project Version** (Object International)	$1,995 $95	IBM PC (DOS/Windows 3.0), OS/2, Mac; Unix version in development. Same as above.	Provides five-layer automation support for OO analysis as described in *Coad/Yourdon Object-Oriented Analysis*, second edition, including: documentation tools — templates, customizable document generation, and a variety of specifications capture points; support for larger models, including collapsible subjects, layer combinations, "hot" overview windows, and views with filters; "on-the-fly" and on-command model critique capabilities; direct connectivity with other tools via a human-readable "SGML" tagged language; windows-and-menus interaction; and several examples to facilitate technology transfer.

Table 4 — Object-Oriented Databases

Name of Product (Vendor)	Price	Hardware (Operating System)	Comments
G-BASE/W (Object Databases, Inc.)	$5,000	Single-user, Lisp-based workstation version; Sun, Apollo, Symbolics, TI MicroExplorer and Explorer, Mac.	Heterogeneous, client-server architecture for CAD/CAM, CIM, CASE, etc., and high-speed transaction processing applications. Features: multiple inheritance for code reusability; abstract data modeling; decision support for "what-if?" scenarios; G-LOGIS, a logic programming facility combining PROLOG and Lisp functions for complete DB inferencing; fault tolerance and simultaneous, on-line backup. In addition, various add-on packages are available, such as G-BASE/SQL, which allows shared, common DB access via an Oracle server.
G-BASE/GTX (Object Databases, Inc.)	Varies by platform	Multi-user, Lisp-based version; Vax/VMS, Sun 3 and 4. C++ version scheduled for early 1992.	
GemStone (Ver. 2.0) (Servio Logic Corp.)	$12,000-$92,000	Sun 3 and 4, DecVAX (VMS), and IBM RS6000 platforms; IBM PC, Mac II, and VT100 workstations. Also supports TCP/IP and DECnet protocols.	Multi-user, client-server architecture for same or heterogenous configurations. Features: ability to store both slot and method descriptions together; data definition and data manipulation languages for supporting applications in C, C++, Smalltalk, and COBOL; graphical tool for creating class definitions. Implementation in X Windows/OSF Motif provides mouse-driven interface for direct manipulation of relationships between classes; user can define classes and their instance variables, create and manipulate class hierarchies, and define sets for holding objects.
IDB Object Database (Persistent Data Systems)	$2,500-$6,000	IBM PC (DOS/Windows), Mac (OS 6.0 and 7.0), NeXT (v. 2.0), Apollo (HP/Domain OS), HP (Unix), and Sun workstations (Unix).	For stand-alone or networked applications; supports controlled sharing of information across heterogenous networks with any mix of supported platforms, including DOS/Windows 3.0, Sun, and HP/Domain OS. Application areas include CASB to advanced office automation, including multimedia.
ITASCA (Itasca Systems Inc.)	$3,995	Apollo, HP, IBM, DEC, Sun, and Silicon Graphics workstations; requires 16 mb RAM.	Based on ORION prototype developed by MCC. Executes routines written in C, Fortran, and Lisp. Applications written in C++, C, or Lisp can call Itasca via interface routines. Persistent objects and methods developed using one language can be called by implementation written in other languages. Features: dynamic schema modification — enables network-wide DB access while changes, such as adding or removing classes, or changing class hierarchies or inheritance, are made at any site; implicit security authorization — users authorized to read data at certain security levels automatically have access to new objects at that level or lower; dynamic binding of version references — allows one object to dynamically "point to" the most recently released or another designated version; support for private and shared DBs — objects may be "checked out" of shared DBs for use in private DBs, where their access then approaches single-user system performance.

Table 4, cont. — Object-Oriented Databases

Name of Product (Vendor)	Price	Hardware (Operating System)	Comments
Objectivity/DB (Objectivity Inc.)	$3,000-$30,000	IBM 386, 486 (DOS and Windows 3.0), Sun 3, SPARCstation (Unix), HP 9000 series (HP-UX), IBM RS/6000, DEC RISC, VAX/Ultrix, VAX/VMS, Sony News, Silicon Graphics workstations.	For applications involving complex data types, relationships, concurrency. Fully distributed architecture with transparent heterogenous support across all platforms. Allows integration of design and language tools for network-based, multi-user applications. C and C++ interface for application developers.
ObjectStore (Object Design)	$25,000	Sun 3 and 4 (Unix); additional platforms/ systems under development: HP, OS/2 (X Windows), and MS Windows 3.0.	Allows integration of existing C libraries and applications with new ones written in C++. Features Schema Designer, an interactive, graphical tool for developing and manipulating large class matrices. Supports source-level debugging and has a graphical, DB browser/query capability for facilitating development. Query optimizer helps minimize number of objects examined in response query. Version and configuration supports allow developers to maintain different versions of a design at the individual object level.
ONTOS DB (Ontos Inc.)	Varies by platform	Sun 3, 4, SPARCstations, Apollo, RS/6000, DECstation (Ultrix), and OS/2 platforms.	For network-based, data-intensive applications ranging from telecommunications and multimedia to manufacturing, geographical information systems, and defense. Allows storage of different types of information: data, text, graphics, pictures, voice, video, etc. Provides interfaces for C++, C, SQL, and DB administration — object language support and graphical utilities for schema design tools — DB Designer, ONTOS SQL, and ONTOS Studio.
Statice (Symbolics, Inc.)	$10,000 Site. $40,000-$50,000	Symbolics machines (Lisp).	Persistent storage allows objects to be stored outside of virtual memory, enabling applications running on different workstations to share the same code. Has methods for attaching procedures to types, thereby associating programs and data. Full commonality with the Genera-Lisp environment, allows integration with other Lisp applications, such as Symbolic's Joshua expert systems programming language.
Versant (Versant Object Technology Corp.)	$15,000	Sun 3 and 4 workstations (Unix), IBM RS 6000.	Intended for application areas such as engineering and office automation, etc. Objects may be created to contain data that can be closely linked to applications; objects (software modules) are reusable and easily modified. Consists of four modules; Versant — a distributed object manager and object storage system for handling objects and performing I/O functions; a C++ class library and set of tools that include a schema utility that allows C++ programs to access a DB; a C library; and Versant View — a development tool for browsing DB objects and classes.

Object-Oriented Product Vendors

Apple Computer
MacApp
20525 Mariani Ave., M/S 33G
Cupertino, CA 95014
(800) 282-2732 or (408) 562-3910

Blaise Computing Inc.
Turbo Vision Development Toolkit
819 Bancroft Way
Berkeley, CA 94710
(800) 333-8087 or (415) 540-1938

Borland International Inc.
ObjectVision
1800 Green Hills Rd.
P.O. Box 660001
Scotts Valley, CA 95067-0001
(800) 331-0877 or (408) 439-4825

Bowers Development Corp.
AppMaker
P.O. Box 9
Lincoln Center, MA 01773
(508) 369-8175

Cadre Technologies
Teamwork
222 Richmond St.
Providence, RI 02903
(401) 351-5950

Chen & Associates
Object Designer
4884 Constitution Ave., Ste. 1E
Baton Rouge, LA 70808
(504) 928-5765

CNS Inc.
C_talk/Views
7909 Shady Oak Rd.
Minneapolis, MN 55344
(612) 944-0170

Coopers & Lybrand
Application Manager
SoftPert Systems Division
One Main St.
Cambridge, MA 02142
(617) 621-3670

Interactive Development Environments
Software Through Pictures
595 Market St., 10th Fl.
San Francisco, CA 94105
(415) 543-0900

Enfin Software Corp.
ENFIN/2
6920 Miramar Rd., Ste. 307
San Diego, CA 92121
(619) 549-6606

Expertelligence Inc.
Action!, SPOKE
5638 Hollister Ave., 3rd Fl.
Goleta, CA 93117
(805) 967-1797

ImageSoft Inc.
CommonView 2
2 Haven Ave.
Port Washington, NY 11050
(800) 245-8840 or (516) 767-2233

Itasca Systems Inc.
Itasca DB
2850 Metro Dr., Ste. 300
Minneapolis, MN 55425
(612) 851-3155

MDBS Inc.
Object/1
Two Executive Dr., P.O. Box 248
Lafayette, IN 47902
(317) 463-4561

National Instruments
LabVIEW
6504 Bridge Point Pkwy.
Austin, TX 78730
(512) 794-0100

NeXT Inc.
NeXtStep
3745 Deer Creek Rd.
Palo Alto, CA 94304
(415) 424-0200

Object Databases, Inc.
G-Base
228 Broadway
Cambridge, MA 02154
(617) 354-4220

Object Design
ObjectStore
One New England Executive Park
Burlington, MA 01803
(617) 270-9797

Object International, Inc.
OOATool
8140 N. MoPac Expwy., 4-200
Austin, TX 78759-8864
(512) 795-0202

Objectivity Inc.
Objectivity/DB
800 El Camino Real, 4th Fl.
Menlo Park, CA 94025
(415) 688-8000

Ontos Inc
ONTOS DB
Three Burlington Woods
Burlington, MA 01803
(617) 272-7110

ParcPlace Systems
Objectworks
1550 Plymouth St.
Mountain View, CA 94043
(415) 691-6700

Persistent Data Systems, Inc.
IDB Object Database
75 West Chapel Ridge Rd.
Pittsburgh, PA 15238
(412) 963-1843

ProtoSoft, Inc.
17629 El Camino Real, Ste. 202
Houston, TX 77058
(713) 480-3233

Quintus Computer Systems
MacDialog Editor
1310 Villa St.
Mountain View, CA 94041
(415) 813-3800

Servio Logic Corp.
GemStone
1420 Harbor Bay Parkway, Ste. 100
Alameda, CA 94501
(415) 748-6200

Symbolics Inc.
Statice
Eight New England Executive Park
Burlington, MA 01803
(617) 221-1000

TGS SYSTEMS
Prograph
2745 Dutch Village Rd., Ste. 200
Halifax, Nova Scotia B3L 4J1,
Canada
(902) 455-4446

Versant Object Technology Corp.
Versant
4500 Bohannon Dr.
Menlo Park, CA 94025
(415) 325-2300

Appendix F:

Object-Oriented Database Products: Status as of November 1991

DOE Contact
Lloyd Arrowood, Martin Marietta Energy Systems; 615:574-8700; arrowoodlf@ornl.gov

The following pages contain a detailed summary of the status of six object-oriented database products that are available in November 1991. The original version of this table was produced in 1990 as part of the NASA-sponsored Advanced Launch System project by Don Rudisill, Bob Owens, and Ann Tipton of Martin Marietta Astronautics Group in Denver. The table was updated in late 1991 by Lloyd Arrowood.

Method of information gathering
For the original version of the table: Each of the products was reviewed, and then information obtained from sales presentations and spending 3 to 5 hours with each vendor was summarized. A composite list of features was created that contained all the features which any of the vendors provided. Each vendor was asked to respond to their capabilities in each of the areas from this composite list of features.

For the revised version of the table: Each product was reviewed and its vendor was interviewed to determine all information which should be changed. All vendors made updates. Thus the following tables do not describe the features of an ideal object-oriented database, but rather they describe the features actually present (or firmly planned) in products as of November 1991.

Products reviewed

Objectivity from:	Objectivity 800 El Camino Real, Suite 200 Menlo Park, CA 94025	Phone 415: 688-8000
Object Store from:	Object Design One New England Executive Park Burlington, MA 01803	Phone 617: 270-9797
Ontos from:	Ontologic Three Burlington Woods Burlington, MA 01803	Phone 617: 272-7110
Versant from:	Versant 4700 Bohannon Drive Menlo Park, CA 94025	Phone 800: 962-5328
GemStone from:	Servio Corporation Suite 110 950 Marina Village Parkway Alameda, CA 94501.	Phone: 510: 814-6200 800: 243-9369
ITASCA from:	ITASCA Systems, Inc. 2850 Metro Drive, Suite 300 Minneapolis, MN 55425	Phone 612: 851-3155

Current release: v1.2
Next release: v1.5 (December 1991)
Contact: Mike Mitrowski
Date contacted: FAX received October 31, 1991. Follow-up on November 1, 1991.

OBJECTIVITY PERFORMANCE

Fetch overhead	Objectivity has designed their database for balanced performance for use in engineering or scientific application areas. That means the database management system performance must be acceptable across several types of users and operations. Additional performance issues considered included: opening a working set of data ("cold start"), opening and modifying a working set larger than allocated swap space, opening several objects from several different databases stored on several different servers, and committing transactions of large amounts of data to several databases.
Query performance	Named and keyed objects supported for fast lookup. B+ Trees supported in Release 2.0. Another significant aspect of performance is whether objects are opened on demand or whether the entire database must be read into memory before searching can begin. Objectivity is extremely fast in accessing objects because they are only opened on demand.
Paging speed	As fast as the disk will allow.
Commit overhead	Worst case overhead is writing dirty pages from the cache to the disk. Shadowing approach is used, which involves moving a pointer from the old data to the new data on the disk. Was essentially instantaneous on a single node.
In memory performance	Memory speed plus the overhead of one test and one indirect pointer.
Tuning statistics	yes
Adaptive	Caching algorithm is adaptive.

OBJECTIVITY SECURITY

Audit Trail	release 2
Access control	C2 security at release 2, intentions to upgrade to B2 later

OBJECTIVITY DATA ORGANIZATION / REPRESENTATION

Versions	linear and branching
Configuration management	Associations support highly complex relationships between objects without database or file boundaries. These relationships are typed, and provide a mechanism for referential integrity of the data. Objectivity/DB provides associations that can be one-to-one, one-to-many, many-to-one and many-to-many. In addition, unidirectional associations are provided for performance. Propagation of operations along these association links between objects is provided. This mechanism is useful for more than just configuration management. Customers establish their own configuration management policies and use versioning primitives within Objectivity/DB to implement those policies.
Nested Transactions	some support in current release
Dynamic arrays	yes with a capacity of 2 gigabytes
Query facilities	Iterators: scan all instances of a class, all objects in a many to many association, all objects at the next level in the hierarchy (contains), all names in a scope. Iterators are not ordered. Can delete current object inside an iteration. SQL where clause in release 2.
collections	NIH library now, more support in release 2.

OBJECTIVITY STORAGE MODEL

Tuneable to application requirements	Cache sizes, page sizes can be controlled. Clustering can be controlled at various levels in the federated data base.
Limits	Federated database capacity $8x10^{18}$ 64K databases / federation 64K containers / database Container capacity 2 gigabytes at Release 1.0 Varrays / object 2 gigabytes at Release 1.0 (no restrictions in size of varray in release 2) 64 bit object ID's, (address space for objects).
Backup/recovery	Backup (incremental at release 2) restore, diagnose, checkpoint, rollback and micro-transactions (undo)
Unix file system	POSIX compatible
Raw device storage	not used for portability reasons

OBJECTIVITY APPLICATION DEVELOPMENT ENVIRONMENT

Schema generation	The DDL preprocessor generates the .h.
Schema evolution	Dump and load utilities can assist. More support in release 2.
Browser	Yes and allows you to monitor locks.
query	yes
change values	yes
who calls	no
who sends	no
class hierarchy	yes
Debugger (source code)	macros for dbx, 3rd party
Application Prototyping	compatible with several 3rd party solutions
form editor	""
graphics editor	"" (ICS)
4th GL	""
Incremental Compiler	Saber C++ highly recommended when available.
Administration tool	In the browser

OBJECTIVITY DISTRIBUTED

2 phase commit	yes
Change notification	release 2
Personal database	yes
Deadlock detection	timeout, wait forever on short transactions in release 2 Notified if the object was checked out for a long transaction.
Locking	at each level of the federated system
Long transactions	yes
Architecture features/limitations	Fully distributed data and control, heterogeneous machine and operation system support via on-demand translations.Supports databases distributed across nodes (multiple servers) with association links across databases. Supports working sets that exceed swap space limitations.
Heterogeneous machines	yes
Relational gateways	future
Long distance slow links	no

OBJECTIVITY OTHER HOSTS

Mac	
Other Workstations	Dec VMS in release 1.1. 3rd major workstation vendor supported in release 2. HP, Sun, IBM, Sony, SGI
Pc	DOS, PS/2

OBJECTIVITY C++ Interface

Virtual function support	yes
Parameterized types	no, as compilers support

Current release: v1.1
Next release: v1.2 (November 1991)
Contact: Frank Broskovetz
Date contacted: November 1, 1991

OBJECT DESIGN PERFORMANCE

Fetch overhead	Unit of fetch is the page or segment as specified by the user.
Query performance	High level of optimization. The compiler generates code for several different strategies, and the appropriate one is executed at runtime.
Paging speed	Normally no pointer update is needed. Pointers must be updated if a collision occurs, due to conflicting assumptions among several open databases about address allocation. Pointers are written back with the updated values to reduce the need for adjustment on the next fetch.
Commit overhead	Cost is just that of writing modified data and the commit mark to journal. DB update is done during idle time.
In memory performance	As fast as dereferencing a C pointer. I was shown preliminary results from running Rick Cattell's benchmarks. Object Store was faster than the predicted optimum on the warm tests, and within about 20-30% of the predicted on the other tests.
Tuning statistics	yes
Adaptive	Adaptive pointer relocation (see paging speed)

OBJECT DESIGN SECURITY

Audi[illegible]l	Versions can be used to track changers of objects. The application would be responsible for tracking who has viewed or read an object.
Access control	Similar to Unix world, group, and user control.

OBJECT DESIGN DATA ORGANIZATION / REPRESENTATION

Versions	Linear and branching. Access to number, or most recent. Compare and merge.
Configuration management	Configuration object. Can propagate operations across relationships through bidirectional pointers.
Nested Transactions	yes
Dynamic arrays	yes
Query facilities	Nested queries containing any condition expressible in C++ are supported.
collections	Are implemented as parameterized types.

OBJECT DESIGN STORAGE MODEL

Tuneable to application requirements	The server parameters file allows changes to size of journal disk buffers and server cache. Cache manager controls the size of local cache on client machines.
Limits	none
Backup/recovery	Import/export facility available.
Unix file system	yes
Raw device storage	Depends upon I/O characteristics, but generally better than Unix file system performance

OBJECT DESIGN APPLICATION DEVELOPMENT ENVIRONMENT

Schema generation	Graphical schema designer which produces C++ code.
Schema evolution	Should be available with release 2.0
Browser	yes
query	yes
change values	no
who calls	no
who sends	no
class hierarchy	yes
Debugger (source code)	yes
Application Prototyping	
form editor	future
graphics editor	future
4th GL	future
Incremental Compiler	no
Administration tool	Complete package of utilities.

OBJECT DESIGN DISTRIBUTED

2 phase commit	yes
Change notification	no
Personal database	yes, optimized for local performance
Deadlock detection	Can predict and stop before timeout.
Locking	Locking is at page level and is transparent to the user.
Long transactions	yes, others can read the previous version until the new version is checked back in
Architecture features/limitations	Multiple server, multiple clients Object design has integrated database technology and the C and C++ programming languages to the extent that most traditional code for accessing the data base is not necessary. Objects are declared as persistent and transaction boundaries are defined directly in the C++ code. Everything else is basically transparent to the application developer. Existing C and C++ libraries can be linked in with no changes.
Heterogeneous machines	1992
Relational links	Release 2.0 or 2.1
Long distance slow links	yes

OBJECT DESIGN OTHER HOSTS

Mac	1992
other workstations	Sun 3s and 4s, RS 6000s, DEC Ultrix, and HP/UX (beta). Others, including DEC VMS and SGI, are planned
Pc	OS/2 in 1992; Windows 3.0 (beta)

OBJECT DESIGN C++ Interface

Virtual function support	complete, no limitations
Parameterized types	yes
Library compatibility	yes

Current release: v2.1
Next release: v2.2 (1st Q 92); v3.0 (late 1992)
Contact: Joshua Duhl
Date contacted: OOPSLA '91

ONTOS PERFORMANCE

Fetch overhead	Objects can be fetched either individually, in aggregate clusters or highly efficient memory managed pages. Fetches are optimized to user RPCs or Inter Process Communication if client and server are on the same host.
Query performance	In a number of independent benchmarks against other Object Databases Ontologic claims that ONTOS has consistently out performed the competition in the amount of time it takes to activate a model from the database. Besides the intrinsic object activation protocol ONTOS supports Object SQL. ONTOS will use indices to provide significant optimizations of this language to further increase ad-hoc query performance.
Paging speed	ONTOS manages client memory to minimize Operating System page faults. Ontologic believes that no single memory management technique is capable of satisfying a diverse range of applications, e.g., applications manipulating very large numbers of small objects will not perform well using techniques that support the management of a smaller number of large objects. Therefore ONTOS permits the selection of appropriate memory manager(s) for optimal application performance.
Commit overhead	ONTOS journals after images. Data modified by the application is written to disk asynchronously immediately following completion of the commit. Consequently the application program experiences no commit delay. Also, by employing memory managers the amount of server processing is minimized due to less need to re-organize the cache.
In memory performance	Reference traversal is as fast as a C language pointer de-reference.
Tuning statistics	In future release.
Adaptive	The availability of a broad range of client memory managers means that an application can be tuned for optimal performance. If the Ontologic supplied managers are not sufficient they can be substituted by user-written methods.

ONTOS SECURITY

Audit trail	A client program can obtain an object from the database which contains the set of all objects which were modified by other clients during a client's own transaction. A transaction audit trail is not planned in the near term.
Access control	Area level security in release 3.0. Also a notion of roles and associated authorizations.

ONTOS DATA ORGANIZATION / REPRESENTATION

Versions	Versions may be included in release 3.0. Both linear and branched versions will be supported, along with the ability to make objects immutable.
Configuration management	Release 3.0 may support configuration management. Primitives will permit objects to maintain their associations with configurations and render configurations immutable.
Nested Transactions	Yes. Serve to isolate client program changes to data.
Dynamic arrays	yes
Query facilities	Objects can be queried using embedded OSQL. The syntax of which is: SELECT object_expression_list FROM aggregate_expression_list WHERE boolean_expression It is possible to specify member functions in any expression. An object expression can contain an optionally fully qualified property name, or a procedure invocation. An aggregate expression can return a type name, a named aggregate, an aggregate value specification, or a procedure returning an aggregate. The boolean expression must evaluate to either a TRUE or FALSE.
collections:	ONTOS supports additional aggregate types besides Dynamic Arrays. These are lists, dictionaries, and sets. These can be either persistent or non-persistent and contain any type of user or system defined object. They use very efficient access mechanisms employing linear hashing and balanced binary trees.

ONTOS STORAGE MODEL

Tuneable to application requirements	Selection of appropriate client memory managers provides an ability to optimize application performance.
Limits	database can hold 2E64 objects Current limits (scheduled to be removed) are: a logical database name, 256 characters database area file name, 80 characters number of servers comprising a database, 80 or more
Unix file system	ONTOS can be fully distributed across a network of platforms. A database is composed of any number of UNIX files spread across the network. Every area is serviced by a ONTOS Server process. An Area can not grow beyond the size of the UNIX disk partition in which it is contained. Areas can be added dynamically to a database.
Raw device storage	None yet

ONTOS APPLICATION DEVELOPMENT ENVIRONMENT

Schema generation	Schemas can be generated in three ways: -Use the Advanced Decision Technology's CASE product called PTECH to graphically describe a conceptual model of the application and generate C++ class definitions and member function code. -Edit a C++ class definition file and "compile" it into the database using the ONTOS utility program called "classify". -Graphically edit the schema using the ONTOS X-windows Motif schema editor.
Schema evolution	Addition of class properties and member functions to Types for which instances exist in the database is fully supported. Instance migration and schema evolution will be part of Release 3.0.
Browser	Yes, and X-windows Motif graphical schema editor.
query	yes
change values	yes
who calls	no, but will support Saber's environment in Release 2.2, which does
who sends	no
class hierarchy	yes, displayed in user-specified graphical formats
Debugger (source code)	An interface to the GDB debugger is provided.
Application Prototyping	Currently can use the PTECH CASE tool to provide the capability to generate application prototypes before generating C++ ONTOS code. Release 2.2 will include Studio I/O, a GUI application development tool.
form editor	In Release 2.2 (see note above)
graphics editor	R2.5
4th GL	Not planned yet (currently in alpha)
Incremental Compiler	Incremental linking technology from Saber is fully compliant with ONTOS (and will be qualifying the ParcPlace incremental compiler).
Administration tool	yes

ONTOS DISTRIBUTED

2 phase commit	yes
Change notification	A changed object set can be iterated over to determine objects changed since the start of a transaction.
Personal database	yes
Deadlock detection	The ONTOS server detects deadlocks and takes appropriate action to rollback the youngest transaction and inform the application. Release 2.2 will have the only distributed deadlock detection of any commercial database product, as far as they know.
Locking	There are 3 types of locks: read, write intent, write. Locks need not be specified. Locks are not directly inherited (but can be programmed) and read locked objects are upgraded to write locked object if they are written. Locks can be placed on individual objects or groups of objects. There's both object-level and page-level locking. ONTOS supports optimistic and pessimistic concurrency control and a third "time-based" model of control. It provides the capability to customize conflict resolution behavior.
Long transactions	yes
Architecture features/limitations	One global, large addressable persistent store spread across several machines. The database owns the areas and servers. No concept of a master database with sub-domains owned by specific users.
Heterogeneous machines	R3.0
Relational gateways	Not yet
Long distance slow links	No Asynchronous client/server communication is available yet.

ONTOS OTHER HOSTS

Workstations	Sun 3, Sun 4, DECstation 3100 and higher, Apollo DN 3XXX and 4XXX, IBM RS6080, OS2. In R2.2 VAX/VMS (single process version). In progress (by Q1 92): SCO UNIX, HP 9000/700 series.
Mac	Waiting for release of MAC OS V7.0.
Pc	OS/2. Intention to port to DOS Windows 3.0.

ONTOS C++ Interface

Virtual function support	Member functions and triggers can be dynamically invoked from a client application through the database without the necessity of being compiled into the application.
Exception handling	Proprietary support of fully integrated exception handling mechanism.
Parameterized types	Available as 3rd party vendors provide support in their compilers. ONTOS supports the ability to dynamically add and maintain types at runtime. Aggregates are parameterized. C++ introduces templates as parameterized types and will be supported.

ONTOS C Interface

Availability	None planned.

ONTOS Smalltalk Interface

Availability	In progress.

Current release: v1.6
Next release: v1.7 (December 1991); v2.0 (June 1992)
Contact: W. Edward White
Date contacted: Telephone conversation on November 5, 1991.

VERSANT PERFORMANCE

Fetch overhead	Uses dual caching. Pages are retrieved from disk on the server, objects are placed in the client cache (client and server can be on the same or different host machine), using virtual memory caching.
Query performance	Pre-fetching and optimized traversal and scanning. B trees and hash tables on attributes. Queries are parsed by a query optimizer to determine the most efficient access path.
Paging speed	
Commit overhead	Using Cattell benchmarks, 20,000 objects were committed in 4.85 seconds on a SparcStation 2 and 2.41 seconds on a RS 6000.
In memory performance	You can refer to the memory location of an object with a pointer or a link (an indirect reference that is converted to a pointer when dereferenced).
Tuning statistics	yes
Adaptive	Intelligent buffer management.

VERSANT SECURITY

Audit trail	Because short and long transactions can be named and logged, database changes can be audited.
Access control	Access control is secured at operating system and database levels using an encrypted password and user name access control list.

VERSANT DATA ORGANIZATION / REPRESENTATION

Versions	Versioning is supported at the object level; versions can be both linear and branching; versions can be accessed by most recent, with or without status, or by version #.
Configuration management	Status is maintained (working, released, transient). Configuration of versions is maintained even when versions evolve across database boundaries if all of the objects in a configuration are checked out in a single request.
Nested Transactions	Short transactions can be nested inside long transactions. Short transactions can not be nested inside other short transactions; savepoints can.
Dynamic arrays	yes
Query facilities	Distributed queries are supported with 2 phase commit across databases. Index support allows speed up. Available from both language interfaces and navigator. Predicate and associative queries can be formed in C++ as a string argument using attribute names, relational operators, and values.
collections	NIH library with fixes; notion of logical containers allow better performance of read/write operations on complex objects.

VERSANT STORAGE MODEL

Tuneable to application requirements	Lots of control on how the data is stored, clustering, buffer sizes, cache sizes. Object cache "tunes" objects retrieved to meet application requests exactly.
Limits	A VERSANT database is a physical storage area that contains the object instances. A database is a set of volumes, not a file, and physically consists of pages, each 16k in size, arranged into extents. When you create a database, you define extent size (default 20 pages). A database can hold up to 2E48 objects plus classes (a class is an object). 2E16 databases an object can be 2E32 bytes
Unix file system	yes
Raw device storage	yes

VERSANT APPLICATION DEVELOPMENT ENVIRONMENT

Schema generation	automatic from .h files
Schema evolution	On leaf classes only for now. Automatic support for addition with default values, and deletion.
Browser	Versant View
query	Versant OSQL
change values	yes
who calls	no
who sends	no
class hierarchy	yes (single and multiple inheritance)
Debugger (source code)	3rd party, including ObjectCenter (formerly Saber C++)
Application Prototyping	Versant Screen
form editor	Versant Screen
graphics editor	Versant Screen
4th GL	Versant Screen
Incremental Compiler	3rd party
Administration tool	yes

VERSANT DISTRIBUTED

2 phase commit	yes
Change notification	You can choose to be notified of certain events. Specifying broken lock notifies you when your soft lock has been broken; request pending notifies you when someone else is waiting for your lock; object available notifies when a reserved object is ready; version creation notifies when a new version is created of an object you are monitoring; object read notifies when an object you are monitoring is accessed; object update notifies when a monitored object is updated; and object deletion notifies when a monitored object is deleted. Notifications are sent in the form of electronic mail.
Personal database	Each workstation can have local personal databases. Concurrency ctrl and logging can be turned off in this database to increase performance.
Deadlock detection	Timeout (notified if a long transaction)
Locking	Read and write locks and nolock option for short transactions. Locks are set at the object level. A write lock on an object implicitly sets intention write lock on class. When objects are checked out in a long transaction, locks persist and are restored during the recovery process if the system is disrupted. Several persistent lock types are supported: read lock, write lock, monitor mode checkout, snapshot mode (will become a new version). Persistent locks can have priorities. Specifying a hard priority prevents the lock being broken by a conflicting request. A soft priority can be broken by a conflicting request for a hard lock. You can specify queuing options in case another user has already locked the object and notifies you when it is available. An object can be checked out of a group database into a personal database. If checked out and without locks, objects can be checked back in as new versions. If checked out with persistent locks, they will be checked in as updates.
Long transactions	yes; checkout copies objects to personal database
Architecture features/limitations	The data base system uses a client-server model in networked computing environments. You can create a group database on a server that can be accessed concurrently by multiple clients. Databases and applications may be developed on a workstation and then migrated to one or more group databases on servers. A workstation can also take on the role of a server. Data can be retrieved from multiple servers. Every object has a logical object identifier (OID). Objects can be moved from one database to another without changing their OID.
Heterogeneous machines	Transaction-level heterogeneity provided.
Relational gateways	Under development.
Long distance slow links	

VERSANT OTHER HOSTS

Mac	no
Pc	Plan to support OS/2 and WINDOWS 3.0
Workstations	Sun 4; RS 6000; HPs, Sequent, DEC (beta)

VERSANT C++ Interface

Virtual function support	yes
Parameterized types	yes

Current release: v2.5
Next release: v3.0
Contact: Adrian Blakey
Date contacted: FAX received November 13, 1991. Follow-up on November 15, 1991.

GEMSTONE PERFORMANCE

Fetch overhead	GemStone provides both remote procedure call application interfaces and a linked application interface. Object transmissions can be bulked. The user can also specify clamps upon a class, instance variables, or specific instances to prevent unwanted data from being automatically transferred to the application.
Query performance	Associative access queries (select:, detect:, reject:, collect:) receive major performance improvements by the creation of indexes. Computationally intensive queries which require examination of disconnected objects (where neither localized navigation nor associative access are possible) can be performed with the speed of a compiled language by adding user-actions written in C to GemStone.
Paging speed	The use of GemStone's clustering mechanism allows the user to co-locate objects in a manner which is appropriate for the commonly performed queries and retrievals.
Commit overhead	In release 2.5, database configuration parameters can significantly improve commit performance.
In memory performance	
Tuning statistics	GemStone provides statistics on individual object size, ownership, class, and access protection as well as the amount of space used by object headers versus data and the number of free pages present within the database.
Adaptive	GemStone automatically modifies the internal storage design of objects as they pass crucial size boundaries, decomposing large objects into trees of smaller objects for fast modification and retrieval.

GEMSTONE SECURITY

Audit trail	GemStone does not currently maintain an audit trail. However, writing a log of object touches is supported by the internal architecture and is used within Servio's engineering lab for internal debugging of GemStone.
Access control	User accounts with password protection for database session login. Object access protection settings of: (none read write) for owner group and world. Object hiding is supported by dynamically specifiable namespaces. The namespaces may be inserted and removed from user accounts at any time and are themselves protected by the access protection settings listed above.

GEMSTONE DATA ORGANIZATION / REPRESENTATION

Versions	GemStone currently allows users to create their own versioning systems on top of Opal, but does not provide kernel level support for versions.
Configuration management	same as versions
Nested Transactions	future
Dynamic arrays	yes
Query facilities	GemStone provides both application programming interface support for embedding queries and database behavior to perform queries. GemStone Forms and GemStone Graphical Query (available to early ship customers), will simplify interactive querying.
collections:	Similar to SmallTalk. Associative access methods are provided for predicate-based element retrieval on sets and bags. Fast set union, intersection, and difference methods are also provided. Indexes based upon element value or nested element value can be dynamically created, queried about, and removed.

GEMSTONE STORAGE MODEL

Tuneable to application requirements	future
Limits	database can store 2 billion objects objects can be up to 1 billion bytes or references to other objects
Unix file system	yes
Raw device storage	yes

GEMSTONE APPLICATION DEVELOPMENT ENVIRONMENT

Schema generation	Visual Schema Designer
Schema evolution	GemStone currently allows classes to be redefined with existing instances able to swap their reference to the new class definition. Additional support for schema evolution is being provided incrementally.
Browser	Inside of GemStone Organizer, a top level 'environment' based on the folder/directory model that allows users access to all tools as well as a structured browser that acts as a catalog of data and application objects.
query	Specification of queries by graphically designating selection criteria in the graphical query tool. The GemStone SmallTalk Interface (GSI) provides a workspace for interactive query execution. The topaz command line interface also has these capabilities.
change values	The GSI provides a window-based instance inspector which allows value modification.
who calls	no
who sends	no
class hierarchy	
Debugger (source code)	yes
Application Prototyping	
form editor	yes
graphics editor	yes
4th GL	OPAL
Incremental Compiler	yes
Administration tool	A database administrator tool which allows the DBA to create and modify user accounts, assign default namespaces and access authorization settings, and specify groups of users.

GEMSTONE DISTRIBUTED

2 phase commit	future
Change notification	future
Personal database	future
Deadlock detection	future
Long transactions	future
Locking mechanisms	GemStone provides 3 types of locks: read (shared), write, and exclusive. In addition, 2 levels of granularity are available (object and collection locks).
Architecture features/limitations	GemStone provides a mechanism for performing methods inside the server process.
Relational gateways	bridge to Sybase, Ingres, Oracle, DB2
Heterogeneous machines	currently supports heterogeneous clients
Long distance slow links	Gemstone supports a remote procedure call interface which can be used today for long distance slow links if used in conjunction with SLIP. Personal Database up and download make sense, but long distance remote disk I/O for lots of small interactive queries may not make sense. One of the advantages of GemStone's architecture is the ability for database processing to take place on a separate CPU from the application. This makes long distance slow link usage more feasible. An application at one end of a slow link can invoke a relatively large amount of processing within GemStone at the other end of the link. Communication volume, and therefore elapsed time, can be reduced by moving functionality to GemStone.

GEMSTONE OTHER HOSTS

Mac	Client support through SmallTalk 80 and V/Mac, Think C
Pc	Client support through SmallTalk 80 and V/286

GEMSTONE SmallTalk Interface

Virtual function support	yes
Parameterized types	yes

GEMSTONE C++ Interface

Description	The C++ interface supports C++ version 2.1 and is implemented as a pre-processor for header files based on standard C++ syntax and can be used with compilers from multiple vendors. A comprehensive, proven class library is provided as part of the interface. The interface supports both dynamic object faulting and explicit object storage and retrieval. The C++ interface is built on top of GemStone's C interface, thereby making the full functionality of the C interface available to the application developer.
Virtual function support	The C++ interface currently doesn't store C++ member functions in GemStone. However, Opal (GemStone) methods can be created, modified, and executed from C++.
Parameterized types	The C++ interface will directly support parameterized types at such time as the C++ language supports them.

Current release: v2.5
Next release: v3.0
Contact: Sandy Miezwa
Date contacted: FAX received October 28, 1991.

ITASCA PERFORMANCE

Fetch overhead	Memory management in ITASCA uses both page and object buffers. ITASCA has a traditional database page buffer scheme that contains pages with multiple object. Desired objects move from the page buffer to an object buffer. It is at the object level that communication occurs to the client whether local or over the network.
Query performance	Query optimization includes determining the query graph traversal order, parallel processing of the query graph branches, and the selection of subquery evaluation sites. Optimization uses statistics on the distribution of data in the database.
Paging speed	Paging is minimized through the use of object buffers at both the client and the server.
Commit overhead	Commit overhead is minimized by using an UNDO log and assuming most actions will commit.
In memory performance	The object buffer then provides ITASCA with enhanced in-memory performance because it contains only frequently referenced objects.
Tuning statistics	ITASCA utilities gather class and attributes statistics for use during query optimization. Class statistics include the number of instances of a class and its subclasses. They also include the number of disk pages containing instances of a class and its subclasses. Attribute statistics include the number of unique values of an attribute for a class and its subclasses, minimum and maximum values of the attributes for a class and its subclasses, the index height, and the number of index pages.
Adaptive	Parallel execution of query graph branches occurs on multiple sites automatically. ITASCA automatically chooses the site with the largest estimated data set as the site to process a subquery or merge partial results. ITASCA supports automatic distribution of code in the form of methods. The schema stored on each site contains all methods. An authorized user can update an algorithm for a method an commit it to the shared database. ITASCA automatically distributes that updated code to each site in the network that has a partition of the shared database. Code distribution will occur even if a site is not working at the time through a spooling mechanism. Methods execute directly on the local machine--only data moves among machines at execution time. No single site acts as a master site, thus ITASCA's architecture has no single point of failure. This is important for maintaining a database system with high availability in a networked workstation environment. ITASCA has neither a central data server nor a central name server.

ITASCA SECURITY

Audit Trail	As of November 1991, ITASCA does not have a built-in audit trail. A customized audit capability can easily be created by refining the system-level methods to capture audit information. These system-level methods include make-object, delete-object, make version, promote, demote, checkin, checkout, and others.
Access control	ITASCA has a sophisticated security authorization technique tied to the class hierarchy. It supports both positive and negative authorizations at any level in the class hierarchy. For example, granting access to all objects but one requires only two authorizations: a global grant followed by a specific denial. Authorization extends to classes, instances of classes, attributes, and methods. Also, inheritance of authorization based on the class hierarchy reduces the work of database administration.

ITASCA DATA ORGANIZATION / REPRESENTATION

Versions	ITASCA supports version control of objects. A new version of an object promotes the original or parent object to restrict further changes to the parent. Promoting an object version to a released status restricts any deletion of the object. ITASCA uses generic versions to dynamically reference the most recent or default version of an object,
Configuration management	ITASCA does not promote a single form of configuration management. It provides versioning, dynamic schema evolution, change notification, and private databases that could be customized into a configuration management suitable to the problem domain. Change notification, in ITASCA is either flag-based or message-based. Flag-based notification will identify an updated object upon querying the object for such information. It is a passive notification scheme. Message-based notification, on the other hand, is an active notification scheme. It will execute a method (or code) upon an update or other change to an object. Such methods can send mail messages or invoke other functions. ITASCA does not need configuration management to track application use of methods. Methods are stored in the database and are bound at run-time. A change to a method does not require the recompilation or relinking of application code.
Nested Transactions	Nested sessions are supported which allows for nested transactions. Sessions can also be shared which results in multiple processes/users sharing locks. Long-duration transactions are supported. Long-duration transactions allow users to check objects out of the shared, distributed database into their private databases. Users can then change the objects in the private databases without affecting the shared database or other users. Then, at any later time, the user can check the updated object or objects back into the shared database.
Dynamic arrays	These are supported

Query facilities	ITASCA supports both interactive and programmatic queries on either the private database, the shared database, or a global query encompassing both the private and shared database. The system can also integrate queries with long-duration transactions. It is possible to specify a *CheckinTree* or *CheckoutTree* as a parameter of the query. Using these parameters will cause the selected objects and related objects to checkout or check-in automatically at the end of the query execution.
Collections:	

ITASCA STORAGE MODEL

Tuneable to application requirements	ITASCA supports physical object movement among distributed sites, indexing and clustering to tune the storage mode. It also allows extension of the database manager through refinement, by class, of system methods such as make-object, delete-object, make-version, promote, checkin, checkout, and others.
Limits	The maximum number of classes in a single distributed database is 4.3 billion. The maximum number of attributes per class is 16.7 million. The maximum number of instances per class is 4.3 billion multiplied by the number of private databases in the system. The maximum number of sites in a single distributed database is 32,767. The maximum number of private databases on all sites in a single distributed database is 32,767. The maximum size of a long-data object (audio, image, text, and others) is the size of a physical disk partition.
Backup/recovery	CPU failure recover is implemented with undo logging. This logging keeps the inverse of changes in an incremental log. ITASCA applies this log to the database at recovery time. Recovery goes backwards from the stat at the time of the failure to a consistent database state. The undo operation will roll back incomplete transactions to get to the end of the last completed transaction. Optional mirror writing of data provides for protection against media failure. Upon a primary or backup disk failure, ITASCA issues a warning and sends electronic mail to the system manager warning of the failure.
Unix file system	The ITASCA database is a UNIX file and uses the file system.
Raw device storage	Not available as of October 1991

ITASCA APPLICATION DEVELOPMENT ENVIRONMENT

Schema generation	The ITASCA Schema Editor allows generation and modification of schema from an OSF/Motif compliant graphical user interface through the use of dialogs. See the next section on schema generation.
Schema evolution	ITASCA supports dynamic schema modification to create a flexible environment for changing or customizing a database system. Authorized users can add and remove attributes or change the subclass/superclass relationship at any time. Authorized users can also add or remove partitions of the shared database at any time. All this can be done interactively without affecting other parts of ITASCA at the time changes occur to the schema. There is no need to "bring the system down" to restructure the database.
Browser	
query	As of November 1991, queries may only be executed at the command line or from within a program.
change values	As of November 1991, values may only be changed at the command line or from within a program
who calls	Assume this refers to a cross reference capability. For applications that use methods, this is not needed for configuration management since ITASCA manages methods as part of the schema and supports late or run-time binding of methods.
who sends	Assume this refers to a cross reference capability. For applications that use methods, this is not needed for configuration management since ITASCA manages methods as part of the schema and supports late or run-time binding of methods.
class hierarchy	The ITASCA Schema Editor allows browsing the class hierarchy. Schema changes may also be made directly from the Schema Editor.
Debugger (source code)	Yes. Use your favorite C debugger for C application code. Methods and LISP application code can use the debugger provided by the underlying Common LISP.
Application Prototyping	
form editor	Not available as of November 1991.
graphics editor	Not available as of November 1991.
4th GL	Not available as of November 1991.
Incremental Compiler	Yes. Methods can be added, removed, or compiled at any time.
DB Administration tools	Graphical tool to perform such functions as disk initialization, disk compression and usage, performance tuning, lock monitoring, and connection monitoring.

ITASCA DISTRIBUTED

2 phase commit	Yes
Change notification	Passive notification assigns a set of events to an instance of a notifiable class. These events can be either update or delete. Should an event occur on the instance, references to it become *reference inconsistent*. Querying checks for this inconsistency. Approving changes returns references to a *reference consistent* state. Active notification assigns a set of events to an instance of a notifiable class. These events can be either update, deletion, making a new version, check-in, or checkout. Should an event occur on the instance, the default is to send electronic mail describing the change and the user who made it. Refining this method and related methods allows alternate actions. Replacement methods can perform completely different operations than sending electronic mail. These methods may perform other operations on the database. In this respect, they behave much like daemons.
Personal database	Multiple private databases can exist at a server site. Data can move between the shared database and private databases. Enhanced locality of reference in the private database converts a possible distributed transaction to a local transaction. This reduces the need for authorization, concurrency control, and locking, resulting in near single-user system performance. Authorization is at the private database level for objects in the private database and not at the object level. A work group may also share a private database. Private databases are part of long-duration transactions.
Deadlock detection	A transaction may wait for a lock to become available until a deadlock is detected. The length of the deadlock time-out can be specified by the user. When a deadlock is detected, control is passed back to the user/application indicating the last action was cancelled. The user/application can decide to re-execute or abort the transaction.
Locking	Pessimistic concurrency control at the object level implements serializability, allowing simultaneous, independent transactions to execute in parallel. This allows transactions to request a lock on concurrent updates on the same objects.

Long transactions	Checking out an object from the shared database to a private database starts a long-duration transaction for that object. Checking the object into the shared database ends the long-duration transaction for the object. Long-duration transactions can last any length of time: minutes, hours, days, weeks, and so on. Any number of short duration transactions can occur between any given checkout and check-in. Any number of checkins can occur after a single checkout. Check-in/checkout can be done with a general query as well as by identifying specific objects. Composite objects check-in/checkout in the same way as simple single single objects. Non-versioned objects physically move between shared and private databases during checkout/check-in. versioned objects have new versions made in the target private database at checkout time leaving the parent object in the shared database. Refinement of check-in/checkout methods on a class by class basis allows additional class or application specific operations.
Architecture features/limitations	The ITASCA database management system has features belonging to any database system. This includes persistent storage for data and schema, concurrency control and locking, transaction management, multiple security levels, and logging and recovery for both CPU and disk media failure. Additional features of ITASCA include long-duration transactions, shared and private databases, distributed version control, distributed transaction management, distributed query management, distributed change notification, object migration, dynamic schema modification, and an extensible architecture. Shared and private databases exist in a distributed environment in ITASCA. The shared database is distributed across workstations (sites) in a network. An ITASCA server controls the partition of the shared database at each site. ITASCA clients provide transparent access to the various partitions of the shared database. The architecture allows any number of private databases at each distributed database site. ITASCA stores the schema redundantly at each site to improve performance, including code in the form of methods. Management of schema updates is automatic for all sites, including sites that were off-line during any changes. ITASCA stores each instance of data in one site. The system or a user may move the data from one site to another. No single site acts as a master site, thus ITASCA's architecture has no single point of failure. This is important for maintaining a database system with high availability in a networked workstation environment. ITASCA has neither a central data server nor a central name server.
Heterogeneous machines	As of November 1991, the distributed system must be a homogeneous set of machines.

Relational gateways	MCC/ITASCA participated in a prototyped interoperability demonstration approved by the SQL Access Group. Remote Data Access (RDA) is the technique used to pass information between clients and servers in a database environment. ITASCA RDA/SQL is not a product at this time, but their participation illustrates the technology that will be applied in the near future (see the July 22, 1991 issue of UNIX Today for more information).
Long distance slow links	Not available as of November 1991.

ITASCA OTHER HOSTS

Mac	Not available as of November 1991.
Other Workstations	Sun-3, Sun-4, Apollo, Hewlett Packard, DECstation, and Silicon Graphics.
PC	Intel 386/486 SCO is scheduled.

Appendix G:

XCUT: AN OBJECT - ORIENTED SYSTEM FOR THE PROCESS PLANNING OF MACHINED PARTS

Allied-Signal Inc., Kansas City Division[3]

Contacts:	Steve Brooks	(816) 997-4329
	Michael Wolf	(816) 997-2999
	FAX:	(816) 997-3331

The Kansas City Division of Allied-Signal Aerospace Company is developing an object-oriented system. XCUT, for the process planning of machined piece parts. The system is currently focused on operation planning for prismatic parts on multi-axis CNC milling machines. The system is expected to reduce flow time by automating time consuming calculations, to reduce cutting tool inventory by presenting appropriate tools for selection from a standard set, and to improve quality through extensive geometry checks and data validation.

A process plan is the sequence of operations necessary to transform raw material into a finished part. XCUT represents a process plan as a sequence of activity objects that are defined recursively. That is, each activity can be a single task to be performed, or a sequence of other activities. This approach allows data to be associated with either an individual task or a group of tasks. For example, an individual task that changes the geometry of the part has an associated solid model, a manufacturing feature, that represents the volume to be added to or removed from the part. Individual tasks may be grouped together according to the resources they are associated with, such as a machine shop, machine tool, part fixture, or cutting tool.

XCUT will be integrated with the business environment around it by direct links to external, relational databases that contain resource information. Those databases will provide the means to validate references in the process plan, such as document numbers, charge numbers and part numbers. It will also permit ad hoc queries into large reservoirs of information such as cutting tool and tooling assembly databases and support stores.

XCUT maintains a persistent definition of products, and process plans in an object-oriented database. The information stored in the database is an implementation of the Product Data Exchange Specification (PDES), an international standard. XCUT incorporates the PDES models for product definition, geometric shape, form features, and process plans, among others. PDES data models specify the definitions of objects as well as an ascii exchange file format for transferring instances of those objects. Each object in the XCUT database has methods for storing and retrieving itself from the database and for reading and writing itself to an exchange file.

XCUT will share its object-oriented database with two other advanced manufacturing projects at the Kansas City Division, the advanced numerical control planning system ANC, and the inspection planning system IPPEX. All three systems will share the same information, providing seamless integration between process planning, NC, and inspection. XCUT will generate process plans which call out tasks for NC analysts and inspection planners. ANC will scan the process plan for NC requests and generate tool paths, which are added to the database. IPPEX will scan the process plan for inspection requests and add inspection tasks to the activity tree.

---

[3] Operated for the United States Department of Energy under Contract No. DE-AC04-76DP00613

Since the object-oriented database used by XCUT is shared with two other projects, the design and implementation of the database is a shared effort. The three development teams use data modeling diagrams to communicate definitions of objects and their relationships and attributes. Once the data models are approved, the diagrams are translated to the EXPRESS language, an ascii representation used by the PDES community to communicate data models. The implementation of the database has been automated by a program that parses the EXPRESS files and generates C++ code and the database schema. The code generated by the parser produces all methods necessary for accessing objects from the database and for importing and exporting objects to data exchange files.

XCUT is linked with a solid modeling system to provide the spatial reasoning capabilities needed in process planning. The solid modeler provides visualization graphics and is used to identify the set of manufacturing features removed form or added to the raw material to make the finished part. The definitions of all solid models used by XCUT are stored in its object-oriented database and are recreated in the solid modeler at run time.

XCUT is being developed on UNIX workstations and is written primarily on C++. It uses Motif and X Windows as its user interface, and PHIGS for its 3D presentation of solid models. It is coupled with the Ontos object-oriented database and the Parasolid solid modeler.

For more information, contact:

S. L. Brooks, M. L. Wolf
Allied - Signal Aerospace Company
Kansas City Division
P. O. Box 419159
Kansas City, MO 64141-6159

Appendix H:

IPPEX: AN AUTOMATED PLANNING AND PROGRAMMING SYSTEM FOR SAMPLE-POINT DIMENSIONAL MEASUREMENT

Allied Signal Inc., Kansas City Division[4]

Contacts:	Curtis Brown	(816) 997-3548
	Michael Wolf	(816) 997-2999
	FAX:	(816) 997-3331

The Kansas City Division of Allied-Signal Inc. is incorporating object-oriented techniques to automate the generation of inspection process plans and part programs for measuring piece parts with coordinate measuring machines (CMMs). The Inspection Planning and Programming EXpert (IPPEX) system requires an integrated environment involving a product definition system, decision making mechanism, resource databases, and a graphical user interface. The system is expected to reduce flow-time for the creation and modification of inspection process definitions, CMM part programs, and support documents. The IPPEX system will also reduce the cost of non-conformance by generating unambiguous instructions, structured inspection plans, and standard and consistent measurement approaches and therefore increase the acceptance of this measurement technology along with increased machine utilization.

Given the inspection scope, IPPEX will plan the sequence of operations necessary to verify that the manufactured part conforms to requirements. These operations will contain activity objects that will identify resources such as measuring machines, pat set-ups, and probe configurations, and tasks such as establish datum reference frame and measure feature. The process plans will also contain inspection techniques based upon the feature's current measurement criteria. The inspection techniques determine the number of sample points, the distribution of these sample points, and the selection of the appropriate substitute geometry algorithm. Based upon this inspection process plan, a Dimensional Measurement Interface Standard (DMIS) formatted CMM part program will be created along with the appropriate part set-up and probe configuration support documents.

IPPEX will interface to external relational databases that contain business information. Those databases will provide the means to validate references in the process plan, such as personnel, document numbers, charge numbers, part numbers, and machine backlog. It will also permit programmatic queries for resource information such as CMM attributes, sensor components, and fixture data.

IPPEX maintains a persistent definition of products and process plans in an object-oriented database. The information stored in the database is an implementation of the Product Data Exchange using STEP (PDES) data models for product definition, geometric shape, form features, tolerances, and process plans.

IPPEX will share its object-oriented database with two other advanced manufacturing projects at the Kansas City Division, the XCUT process planning system for machining, and the advanced numerical control system ANC. All three systems will share the same information, providing

---

[4] Operated for the United States Department of Energy under Contract No. DE-AC04-76DP00613.

seamless integration between process planning, NC, and inspection. XCUT will generate process plans which call out tasks for NC analysts and inspection planners. ANC will scan the process plan for NC requests and generate tool paths, which are added to the database. IPPEX will scan the process plan for inspection points for feature verification or part acceptance and add inspection tasks to the activity tree.

IPPEX implements a product modeling environment which encompasses a feature-based tolerance modeler, for representing explicit geometric dimensions & tolerances, integrated with a commercially available solid geometric modeler. The product modeler provides visualization graphics, associates tolerance features with geometry, and provides spatial reasoning capabilities such as datum reference frames, optimal inspection sequencing, and sensor movement. The definitions of the solid nominal geometry along with its tolerance information used by IPPEX are stored in its object-oriented database and are recreated in the product modeler at run time.

IPPEX is evolving from its conceptual prototype into an object-oriented system. It is being developed on UNIX workstations and primarily using the C++ language. It uses Motif and X Windows as its user interface, and PHIGS for its 3D presentation of solid models. It is coupled with the Ontos object-oriented database, the Parasolid solid modeler, and the CLIPS expert system tool.

For more information, contact:

C. W. Brown & M. L. Wolf
Allied Signal Inc.
Kansas City Division
P. O. Box 419159
Kansas City, MO 64141-6159

Appendix I:

STEP "HAPPENS"[5]

Allied-Signal Inc., Kansas City Division[6]

Contact: John Zimmerman (816) 997-2932
Noel Christensen (816) 997-3984

A true object-oriented STEP prototype implementation project has met its first major milestone. Allied-Signal Inc., Kansas City Division, under prime contract to the U.S. Department of Energy, has reached this milestone with only one-and-a-half staff years of effort.

John Zimmerman and Noel Christensen undertook this translation project when their management indicated they were interested in more than just "models hanging on the walls." As Zimmerman notes, management wanted to "see STEP happen." So they "made STEP happen." They put STEP to work demonstrating a production- quality solid model transfer. An object-oriented transfer, no less.

The Advanced Manufacturing Development System (AMDS) project reached its first milestone by exchanging a solid model between dissimilar solid modelers, then generating production-quality NC cutter paths from the transferred data.

The goal of the AMDS project is to develop a next-generation STEP part database and application environment to produce advanced manufacturing applications. (See Figure 1.) The project seeks to demonstrate STEP file transfer and STEP database usage, support advanced numerical control (NC) and inspection applications, and improve STEP by providing feedback from experience.

AMDS has demonstrated that it can transfer a STEP boundary representation part from the SDRC solid modeler GEOMOD to the object- oriented database (OODBMS) ITASCA (from a company of the same name), then to Spatial Technology's solid modeler ACIS, where it is used with Spatial Technology's STRATA milling package to create production-quality NC cutter paths.

This demonstration was accomplished using independently developed STEP environments at SDRC and Allied-Signal Kansas City.

DEVELOPMENT PROCESS

Chia-Hui Shih at SDRC developed a prototype STEP environment in which she used a relational database (SDRC's PEARL) as an intermediate form. She generated translation routines directly from the EXPRESS STEP definitions. GEOMOD is B-spline-based, so the B- spline data was translated into STEP analytical surfaces (planes and cylinders) and then into a STEP physical file.

---

[5] This article originally appeared in *Product Data International* (ISSN 1050-7043), vol. 2, no. 5, September 1991. Used with permission of Barbara D. Warthen, Editor, Warthen Communications, N5303 Broughton Road, Albany, WI 53502-9725. Phone & FAX: (608) 862-1702.

[6] Operated for the United States Department of Energy by Allied-Signal Inc., Kansas City Division, under contract number DE-AC04-76-DP00613.

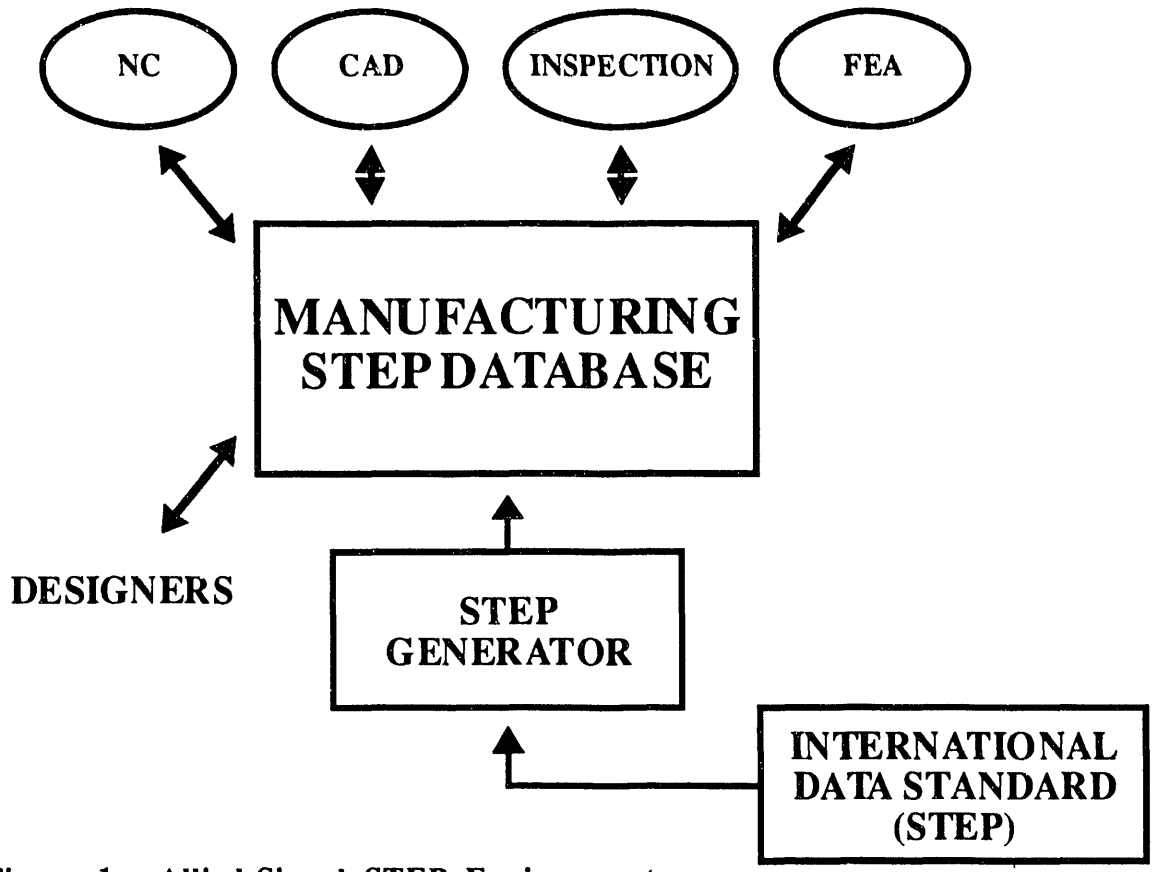


Figure 1. Allied-Signal STEP Environment.

In Kansas City, Christensen developed a prototype STEP environment which uses an object-oriented database (the ITASCA OODBMS) as an intermediate form. He also generated database definitions and translation routines directly from EXPRESS. Christensen received STEP physcial files from SDRC and translated them into ITASCA. ITASCA became home base for the STEP data. Christensen then hand coded a custom translator to move the STEP data from ITASCA to the Spatial Technology ACIS solid modeler. Once in ACIS, the Spatial Technology STRATA milling package was used to generate production quality NC cutter paths from the solid model.

PROGRESSIVE

The Allied-Signal project is progressive. Christensen achieved maximum productivity by programming in the Intellicorp KEE and ITASCA object-oriented languages. He used these high-productivity languages to write software that generates STEP flat file pre- and post-processors and database schemas directly from the STEP EXPRESS definitions. He did not need to embed database calls in his programs because no database calls existed: the ITASCA database management system, for all practical purposes, is invisible to the programmer.

Zimmerman notes that the pre- and post-processors (application programs) are stored in the ITASCA database. Confusing? Traditionally, databases store data. Application programs - including pre- and post-processors - are usually stored separately from databases, the repositories. Zimmerman points to the object- oriented paradigm, which allows information to be stored within (encapsulated in) objects. The information can be methods, which can be the procedures that

traditionally make up application programs. So the programs (the pre- and post-processors in this case) can be embedded in the database objects.

VALUE

Is this of any value - or is it merely the "latest and greatest" new technology? Definitely the former, Zimmerman asserts. By storing methods within objects, synchronization problems disappear, for example. The correct version of the processors and database always exist, because they are all generated from one source and are one unit.

The AMDS demonstration was a STEP Level 2 transfer - but a
Level 2 which is a stepping stone to a Level 3. (Level 1 is a flat file exchange. Level 2 uses a database as a temporary working form. In Level 3 the database is permanent and is used by multiple applications.)

This project is, Zimmerman and Christensen reiterate, an internal development project. Its goal is to "demonstrate STEP capability."

Did they succeed?

"We made STEP happen! Our management saw STEP actually being used to generate production-quality data. This one demonstration did more to remove the mystery from STEP than three years of modeling." A demonstration transfer involving Level 2 and object-oriented databases, no less.

ON-GOING ACTIVITIES

Zimmerman notes that until fairly recently, solids were seen as the basis of CAD/CAM system. Now features are becoming more prominent. Allied-Signal Kansas City Division, through the AMDS project and its involvement in STEP, is now placing more emphasis on features.

AMDS continues to expand. Developers plan to support four major new application areas:
- o generative inspection planning
- o generative process planning
- o feature-based tolerancing
- o constraint-driven geometry creation

This suite of applications is to be supported from one integrated STEP-compatible database.

BENEFITS

Zimmerman comments,

> This project increased our confidence in STEP and gave us something tangible to show our management. We discovered the advantages of using high-productivity programming and the object-oriented paradigm to implement STEP. We reviewed the STEP models more critically than we ever had before. The greatest benefit coming from the project so far has been the acceptance of STEP by the developers of advanced manufacturing applications at our plant. These developers are now asking for more STEP database capabilities than we can deliver!"

Zimmerman and Christensen are both active members of the IGES/PDES Organization.

Project Name: Advanced Manufacturing Development System

Customer: Department of Energy, Process Development Order 70581900

Major project milestones:

FY90:	Technology selection
FY91:	STEP-based solid model transfer
FY92:	STEP-based form feature transfer and integration with USAF Rapid Design System (RDS)

Object Products used:
- Intellicorp KEE: Object-based knowledge shell
- ITASCA: Object-oriented database management system (OODBMS)

Appendix J:

Rapid Response Manufacturing (RRM)

Martin Marietta Energy Systems - Oak Ridge

Contact: William D. Cain, Engineering, cainwd@ornl.gov, 615: 574-3235

Project Sponsor:	DOE - TCI funding through a CRADA agreement with NCMS
Project Type:	Research, Development and Deployment
Start Date:	July 1, 1992
End Date:	July 1, 1997
Manpower Effort:	Y-12 will provide 6-8 FTEs/yr for 5 years; NCMS Joint Venture Companies will provide an average of 30 FTEs/yr for 5 years

Project Overview:

MMES is in the process of establishing a Cooperative Research and Development Agreement (CRADA) with the National Center of Manufacturing Sciences (NCMS) to participate in its joint venture Rapid Response Manufacturing (RRM) Project. The RRM Program is an effort to enable engineers to effectively reduce the design and fabrication time needed to manufacture products rapidly in response to market demand. The objective of RRM is to develop an advanced systems environment which will enable engineers to more rapidly design and manufacture machined parts. A five year program is planned. The technical emphasis will be on feature-based product modeling and knowledge-based applications as a means of attaining rapid response manufacturing. Insofar as possible, RRM is to be built on a foundation of commercially available capabilities such as solids modeling, knowledge-based shells, and database systems. The focus on the program is on designing and making products of faster, better, and cheaper through application of these technologies.

The manufacturing companies involved in this NCMS managed joint venture are Ford, GM, TI, and United Technologies. Vendor companies include Aries, Cimflex Teknowledge, Cimplex Corporation, ICAD, Parametric Technology Corporation, and Spatial Technology. Partial funding for this effort will be provided by the National Institute of Standards and Technology (NIST) through Advanced Technology Program (ATP) funding. The total project cost for the five years is budgeted for about $50 million.

Project Description:

RRM will be accomplished by coordinating and extending the application of integrated product and process modeling, knowledge-based applications, and direct manufacturing in a cooperative environment. Each participant will measure progress relative to the following seven system capabilities that will reduce design-manufacturing cycle time, improve the quality-to-cost ratio,and improve reliability:

* Establishing complete models of design and process data.
* Improving access to product and process knowledge.
* Accurately producing the first part.
* Developing products in a single iteration.
* Developing portability of product models among manufactures.
* Creating new designs from mathematical variation of proven designs.
* Manufacturing parts directly from design models.

Each manufacturing firm has selected a different product family for development. MMES will focus on turned parts and milled/drilled parts.

A major part of the MMES emphasis will focus on establishing a sharable product and process database to support the various diverse applications for design and manufacturing. The reason for this is that currently product data is being maintained in several disjoint poorly conceived databases and presentation formats. Frequently the only access to product data is a paper copy of an engineering drawing. The only electronic representation available is wrapped up in an inaccessible proprietary CAD vendor's database. Manufacturing data is similarly maintained. Product data is transferred between applications through human interpretation of paper drawings or CRT displays. This project will demonstrate that there exists a neutral conceptual schema for product information which is implementation independent and capable of supporting the various applications which use or create it. The emphasis will be on the creation of a product model where the engineering drawing is viewed as a method of presenting product data. The implementation will be object oriented and will conform to emerging standards (STEP) as much as feasible.

Initial Work:

One of the goals of Martin Marietta Energy Systems is to manufacture higher quality less expensive parts faster. To achieve this, engineering, planning, manufacturing and quality functions must be integrated in a seamless environment. This requires developing a highly sophisticated, flexible, complete and reliable product and process data information system. The initial work in this area will be a development effort to provide a base-line product and process data base system utilizing current technologies for the communication of product definition data between dissimilar design and manufacturing applications. This base-line database will utilize existing STEP models wherever possible.

In particular, the initial effort will involve:

1. creating a minimally redundant product data information schema which is application and implementation independent. The latest work within STEP will utilized wherever applicable.

2. the use of object-orient data base and programming techniques.

3. supporting the product definition information requirements of at least two non-homogenous manufacturing applications which share common product data.

4. demonstrating the creation, maintenance and use of electronic product design data.

5. providing test information feedback to STEP

6. identifying incomplete or missing concepts and capabilities within existing vendor supported software. Particular emphasis will be placed on the identification of incompatibilities (both conceptual and application specific) which prevent data integration.

7. recommendations for implementing this system in a production environment.

Appendix K:

DATABASE MODERNIZATION STUDY FOR STRATEGIC COMMAND, OMAHA

Sandia National Labs, Albuquerque

Contact: Laurence Phillips (505) 844-7332

This study is being conducted to determine the advantages and disadvantages of modernizing the databases used during the development of large scale plans. The impetus of the study came from a desire to streamline the planning process. Our thesis is that moving to an environment that permits steps in the planning process to be executed concurrently would give the greatest gain, and that the enabling technology is modern databases.

Our concept of operations is that planning is analogous to specifying, designing, manufacturing, and producing a part: Although the steps in the two processes are different, the sequential value-added approach is the same. Both processes are described in terms of steps, each of which must be completed before the next can begin. Errors that require rework in any earlier steps halt current operations while all data is fed back for correction, then reworked through all intervening steps back to the error detection point.

Correcting this state requires a simultaneous view of the thing being designed by all the enterprises involved in the process so that errors or non-cost-effective decisions can be detected by downstream processes as they are made. The core requirement in operationalizing concurrency is a data communication standard that allows the different operational enterprises to view the data simultaneously.

We are currently testing the hypothesis that storing planning data in a more modern extended relational or object-oriented database would not only allow concurrent planning operations but would also greatly reduce database administration, maintenance, and quality control workloads. We are using the Statice object-oriented database from Symbolics to explore the implementation and concurrent use of actual planning data in a modern data environment. Initial experimentation of the object-oriented approach using Intellicorp's KEE product showed that the approach was feasible. Important attributes of the research system include attribute-level versioning and (promised) portability to Sun/Unix workstations. The study is projected to run through the end of FY94.

For further information, contact:
Laurence R. Phillips
Division 6601
Sandia National Labs
Albuquerque, NM 87185

Appendix L:

"INTELLIGENT" PROCESS CONTROL

Los Alamos National Laboratory

Contact: John Marinuzzi (505) 667-8254

THE CHALLENGE

Los Alamos activities require the operation of many complex chemical processing and manufacturing systems. We are continually trying to improve our operations for our own benefit and because the DOE "modernization" program calls for Los Alamos (and all other facilities) to strive toward the "factory/laboratory of the future" and make major gains in regard to productivity, quality control, waste minimization, safety, flexibility, and cost effectiveness. One promising approach to improve- ment in all these areas is the application of Artificial Intelligence technology to the program of building automatic, "intelligent," process control systems.

THE APPROACH

The historical procedural approach to computer modeling coupled with classical or modern control practice has resulted in extremely limited "intelligent machine systems." Such systems show little or no ability to change or adapt their behavior in response to complex process changes. Moreover, development , modification and maintenance of these conventional software systems is prohibitively difficult.

During the last seven years, Los Alamos has invested millions of dollars in developing Neutral Network/Artificial Intelligence capability at the Laboratory. This technology, which is based upon Object Oriented Programming techniques, has been significantly improved in the last few years in terms of capability, equipment, cost, and commercial availability. We are extending this technology to develop a very high level modeling and process control system for use at Los Alamos.

Neutral networks are being used to recognize and abstract knowledge from complex data. More conventional Artificial Intelligence techniques, based upon object oriented programming, are then used to model this and other knowledge into an "intelligent" control system. Los Alamos has been developing this control system for two years. Some of this technology is mature enough for production use and promises significant gains in productivity. Commercial products from Gensym Corporation, Intellicorp, and Sun Microsystems are used in this effort.

IMPORTANCE TO LOS ALAMOS, THE DOE, AND COMMERCIAL PROGRAMS

Our intelligent process control program is directed toward Los Alamos activities. We expect to achieve significant process improvements in areas of safety, inventory control, quality, production, and costs. However, many elements of this intelligent machine control work are generic in nature and are immediately applicable to other Laboratory man/machine process control and manufacturing programs. Some of these techniques can also be readily applied to commercial applications.

For more information, contact:

John Marinuzzi
Los Alamos National Laboratory
Knowledge Systems Laboratory
MEE-3, MS J580
Los Alamos, NM 87545

Appendix M:

KOALAS: An Architecture for Intelligent Control Systems

Los Alamos National Laboratory

Contacts:
Christopher Barrett, Analysis Division / Simulation Applications
(505) 665-3405 email: cbarrett@lanl.gov

Kathryn Berkbigler, Computer Research and Applications
(505) 667-8377 email: kpb@lanl.gov

Description:

KOALAS is an architectural approach to the design of intelligent control systems that combines more conventional control theory with artificial intelligence techniques for automated reasoning. It structures the process of abductive reasoning for state estimation in complicated environments and separates state estimation from control reasoning processes. It is unique in that it explicitly isolates deductive functions, which are best performed by computers, from inductive functions, which are often best performed by human operators. Moreover it structurally defines the relationships between deductive inference, inductive inference, and hypothesis testing in the control system. The components of the architecture include a sensor data manager which performs hypothesis testing functions such as multi-sensor data association, a simulation module for predicting future system state, an evidence manager which merges all the available evidence and adds operational constraints, and an advice generator which recommends control actions to the operator.

The architecture also includes a role for the human operator as an interpretive, inductive component that is involved in the process of using sensor data to form the situation assessment. The operator may supply hypotheses which can be used to focus the attention of the sensors and reasoning system on areas where more information is needed. The human additionally can perform a supervisory control function, accepting or rejecting any recommended control actions or assumed state.

As an integral part of the design process, our focus is on the modeling and simulation of the various components that will comprise the fielded control system. We use object-oriented techniques throughout.

We are currently using the ProKappa development environment from Intellicorp to support some of our object-oriented software development and for the rule-based expert system used in the advice generator module. The KOALAS architecture supports multiple interacting intelligent objects whose implementations may distributed across multiple platforms, so distributed object-oriented computer simulation is required.

The KOALAS architecture is currently being applied on several projects, including automated multi-sensor integration in tactical naval aircraft. In these projects we are developing techniques for achieving coordinated action by multiple vehicles using data fusion techniques with distributed intelligent controllers sharing only state information.

For more information, contact:

Christopher Barrett
MS F606

Kathryn Berkbigler
MS B265

Both at: Los Alamos National Laboratory
Los Alamos, NM 87545

Appendix N:

GOOSE: A Generalized Object-Oriented Simulation Environment for Developing Dynamic Models

Martin Marietta Energy Systems - Oak Ridge

Contact: Delphy Nypaver, 615:574-2969

GOOSE, a software prototype for a fully interactive, generalized, object-oriented simulation environment, is being developed at the Oak Ridge National Laboratory. Dynamic models may be easily constructed and tested; fully interactive capabilities allow the use to alter model parameters and complexity without recompilation. This environment provides access to powerful tools, such as numerical integration packages, graphic displays and online help. Portability has been an important design goal; the environment, written in Objective-C[7], was originally developed on a UNIX platform and was easily ported to personal computers. GOOSE version 1.4 introduces new enhancements, such as the capability of creating "initial", "dynamic", and "whendo" methods. The concept of modularity used in other advanced simulation packages such as ACSL[8] allows one to build a complex model easily from a collection of previously written components. The object-oriented approach extends this idea to take full advantage of component definitions, allowing precompilation, optimization, and efficient testing and validation of individual modules. Once a library of components has been defined and compiled, system models can be built and freely modified without recompilation.

This is an ongoing effort to create a software library of nuclear reactor component models for the new simulation environment. The most important characteristics of this library are: 1) modularity, with different levels of model complexity, and 2) generality, so that different reactor designs can be simulated. Control algorithm and human models will be included. The models created are being tested and validated against other codes or plant data whenever they are available.

Other contacts:

S. J. Ball
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, Tennessee 37831-6010

L. Guimaraes and M. Abdalla
The University of Tennessee
Department of Nuclear Engineering
Knoxville, Tennessee 37996-2300

Reference

Ford, C. Ed, Carlos March-Leuba, Lamartine Guimaraes, and Daniele Ugolini, "GOOSE, a Generalized Object-Oriented Simulation Environment for Developing and Testing Reactor Models and Control Strategies", *Proceedings of the AI'91 Frontiers in Innovative Computing for the Nuclear Industry*, pp. 694-704, September, 1991.

---

7 The Stepstone Corporation, *Objective-C Compiler Version 4.3 User Reference Manual* (1991).

8 Mitchell and Gauthier Associates, *Advanced Continuous Simulation Language (ACSL) Reference Manual 10.0* (1991).

Appendix O:

ADVANCED SCIENTIFIC COMPUTING ENVIRONMENT TEAM

J. P. Church, Advisory Engineer
Scientific Computations Section, Savannah River Technology Center
Westinghouse Savannah River Company, Aiken, SC 29808

I. OBJECTIVE

The mission of the ASCENT (Advanced Scientific Computing EnvironmenT) Team is to continually keep pace with, evaluate, and select emerging computing technologies to define and implement prototypic scientific computing environments that maximize the ability of scientists and engineers to manage scientific data. These environments are to be implemented in a manner consistent with the site computing architecture and standards and strategic plans for scientific computing.

A broad, long term, goal of the ASCENT Team is to provide a computing environment that will let scientists and engineers function at the higher level of abstraction that is their actual area of technical expertise. The scientist/engineer should be able to solve problems by interacting with conceptual representations drawn directly from the scientific and engineering domains. In this environment the scientist/engineer (i.e., the "problem solver") builds the problem model with reusable virtual objects having associated attributes and behaviors, including any real or artificial constraints. The problem solver could then test the model by perturbing it interactively and observing quantitative (archived experimental measurements; simulated or computed data) and/or qualitative (trends, approximations) responses. Such an environment would greatly aid the solution and understanding of scientific and engineering problems.

Some specific examples may help clarify these ideas. A thermalhydraulics analyst should be able to interact with his[9] desktop terminal to build a RELAP model of a Savannah River Site (SRS) reactor by dragging icons of pumps, pipes, vessels, heat structures, etc., to assemble the completed model. The icons would contain the data, correlations, relationships, physics, etc. pertaining to the actual object they represent. The analyst could then specify a flow transient that reproduces the flow changes imposed during an actual reactor test, compute the results, call up the archived experimental results and display them alongside the computed results, execute other approximate models for the same conditions (e.g., FLOOD code), and display those results to compare with the RELAP and experimental values. The analyst should be able to easily compute correlations and then display and compare trends of both experimental and computed results. Similar analyses could be performed using various reactor physics and charge design codes, Reactor Monitoring System data or Control Computer data, and simulator response.

Another example might be an engineer who needs to know the location of the safety rods in one of the SRS reactors. He would be able to interact with his desktop terminal to request a representation of a reactor facemap, choose a pull-down menu to select safety rods, and request a printout and/or screen display of the X-Y coordinates of all, or any subset of, the safety rods. If the engineer were analyzing the risk of safety rod failure, he could select a number of rods from the

---

9 Throughout this document the use of words denoting gender is for convenience only. The reader should substitute the concept of "the problem solver" in such usage.

facemap display, either singly or by drawing a shape around a group, and choose a pull-down menu to indicate that those rods were assigned 'inoperable' condition. The engineer could then use screen icons representing computer codes and databases to build a procedure that would determine the reactivity worth of the perturbed safety rod complement, and, depending upon the series of codes and recursions specified, could complete the probabilistic analysis of risk. The engineer could interrogate the meteorological database, display a windrose for each stability class, use the mouse to select and perturb a portion of any windrose, and re-do the analysis. Alternatively, the engineer could interrogate the real-time wind data being gathered by the onsite weather towers, and examine alternate evacuation plans by selecting highways with associated attributes of carrying capacity (population source depletion capacity) to minimize both population and maximum individual dose following a postulated major core melt accident.

II. PROPOSED SYSTEM

The major trends in computing hardware and software technology clearly indicate that the future "computer" will be a network environment that comprises supercomputers, graphics boxes, mainframes, clusters, workstations, terminals, and microcomputers (i.e., a full complement of clients and servers). This "network computer" will have an architecturally transparent operating system allowing the applications code to run on any box(es) supplying the required computing resources (e.g., cycles, storage). The environment will include a distributed database and database managing system(s) that permits use of relational, hierarchical, object oriented, GIS, et al, databases.

The benefit of this proposed environment is that it will provide full flexibility to take advantage of the latest advances in hardware and software and, at the same time, maximize the ability to process scientific information and minimize the time required to develop products. And the newly hired S/E will be able to contribute much more quickly to SRS research and development programs.

III. PROGRAM

To reach this long term goal we have implemented a stepwise progression from the present assemblage of monolithic applications codes running on disparate hardware platforms and operating systems.

The initial components of this program include:

- Development of a prototype distributed computing system based on Unix, X Windows, network computing hardware (heterogeneous environment), distributed databases, and distributed file systems.
- Development of portable graphics tools
- Initiation of training to implement and disseminate above tools and methods.

The first item and second items are well underway with significant new capabilities already provided to the applications community onsite. Portable graphics tools are being developed using used Object Oriented Technology concepts and are the subject of this Appendix. More complete information about our program has been published in external reports.[10,11]

---

10 J. P. Church. "Advanced Scientific Computing Environment Group-New Scientific Database Management Task-Program Plan (U)". WSRC-TR-91-70 (February, 1991).

11 J. P. Church. "Progress Report: Advanced Scientific Computing Environment Group-New Scientific Database Management Task (U)". WSRC-TR-91-420 (June, 1991).

IV. GRAPHICS TOOLS AND APPLICATIONS

As discussed above, the ASCENT Team is developing a core set of graphics tools to be used in the scientific computing arena at SRS. These tools are intended to make application development easier, human interfaces more intuitive, and application codes more portable by separating the calculations from their input and output. The graphics tools are being developed using industry standards such as the C Language, X Window System, X Toolkit, Motif™ Graphic Tool Kit, and Unix. Each tool is intended to be the standard graphical user interface for the site and to provide the capability for applications output to be viewed from anywhere onsite. The tools are being developed using Object Oriented Technology concepts.

The reactor facemap tool is discussed next. This tool was used in the subsequent development of a Reactor Monitoring System application (to display online reactor data) and a Graphic Reactor FaceMap function (to create input for charge design) which are also discussed below.

IV.1 *FaceMap Tool*

The first graphic tool completed, FaceMap, displays a reactor facemap[12]. The FaceMap tool has been tested with certified Senior Reactor Operators to identify 'human engineering discrepancies'. The prototype was developed using the Motif™ Graphic Tool Kit, but the production version of the tool was written as a widget based on the Xtoolkit Intrinsics . This facilitated packaging the tool as a separate reusable entity that is distinct from the application.

IV.1.1 *FaceMap Features*

The FaceMap tool provides 'pointer tracking' and 'enter notify' capabilities. 'Pointer tracking' places a crosshair on the facemap at the center of the hex that the mouse pointer is in. The crosshair runs the length and width of the facemap. This makes it easier to determine which x,y location the pointer is in. Pointer tracking also highlights the particular hex by drawing a line around the outside of the hex. 'Enter notify' notifies the application when the mouse pointer moves to a new hex position. An example of a use of this feature would be an application which displays the online computer number corresponding to the position of the pointer.

The tool can outline portions of the reactor facemap (positions, clusters, gangs, sectors and systems), display the facemap in grayscale as well as color, and produce PostScript output for printing the facemap. The facemap will also print on an inexpensive black-&-white PostScript printer and will simulate grayscale.

Presently, the FaceMap tool runs on IBM RS/6000, DEC RISC, HP 700, and Sun SPARC workstations running their individual versions of Unix; VAX systems operating with VMS; and a Mac IIci with A/UX (the Mac X Window System must also be installed). FaceMap can be displayed on any box (including Mac's and PC's) having X server software.

IV.2 *FaceMap Tool Applications*

IV.2.1 *Reactor Monitoring System*

The Reactor Monitoring System (RM, RMS) facilitates monitoring and interpreting reactor operation by providing for collection, storage, and retrieval of reactor operating data. The FaceMap-RMS application prototype provides a full graphic display of a facemap of temperatures,

---

[12] J. C. Roberts. "Interactive Graphical Reactor Facemap Tool. Patent Disclosure No. SRS-91-230 (May 23, 1991).

flows, powers, or any other dimension of the reactor assemblies. This application was extended to display a sequence of historical data at various frame rates. The result is that the display of online reactor data can be viewed from anywhere on the local area network.

IV.2.2 *Graphic Reactor FaceMap (FM)Function*

The latest FaceMap tool application developed is intended to replace the text-based RM function used to facilitate data input to a reactor charge design code. The RM function creates an image of a reactor facemap during charge design. That image is then processed to point to desired data records to describe a specific reactor charge for physics codes calculations. The text-based RM function is a pseudo-graphic that is difficult to use, imposes unnecessary constraints on the user, and is prone to user error.

The new graphics function, named FM, replaces the RM function. Although the RM function will continue to be available, it is expected that user productivity obtained with the new FM function will be so greatly increased that a user will be able to set up a charge design much more rapidly, by a factor of 10-20 for simple problems and as much as 100 times faster for very complicated problems, than previously possible. Early tests show that goal is achievable.

The new function permits multiple axial-level reactor maps, creation of assembly types through an assembly editor, mapping between 2-character mnemonic label and corresponding reactor input data record, cutting and pasting between axial levels, and cutting and pasting of positions, clusters, sectors, systems and gangs.

The new FM function has five panels controlled by the user: (1) FaceMap Panel to display the facemap and status information; (2) Assembly Palette Panel to display a list of the current assemblies that can be put into positions on the facemap; (3) Assembly Editor Panel to create assemblies by specifying a name, type, mnemonic, color, and GLASS record; (4) Preferences Panel to specify if boundaries (gang,sector, or system) are to be shown, if hexes are to be outlined, if assemblies are to be labeled (OLC, mnemonic, or type), if the pointer is to be to tracked with crosshairs and hex highlighting, etc.; (5) Selector Panel to identify specific groupings of reactor positions.

These panels are discussed more fully below.

IV.2.2.1 *FaceMap Panel*

The FaceMap Panel, shown in Figure 1, is the main working area for the FM function. It is used to display the current charge, select positions and it contains the application menus. Shown in Figure 1 is a fictitious example charge with the options menu activated. When a user presses the mouse button on a particular position, that position becomes selected. A selected position will have a white outline around it. Once the position is selected the user may place an assembly into that position or perform an Editing function such as Cut, Copy and Paste.

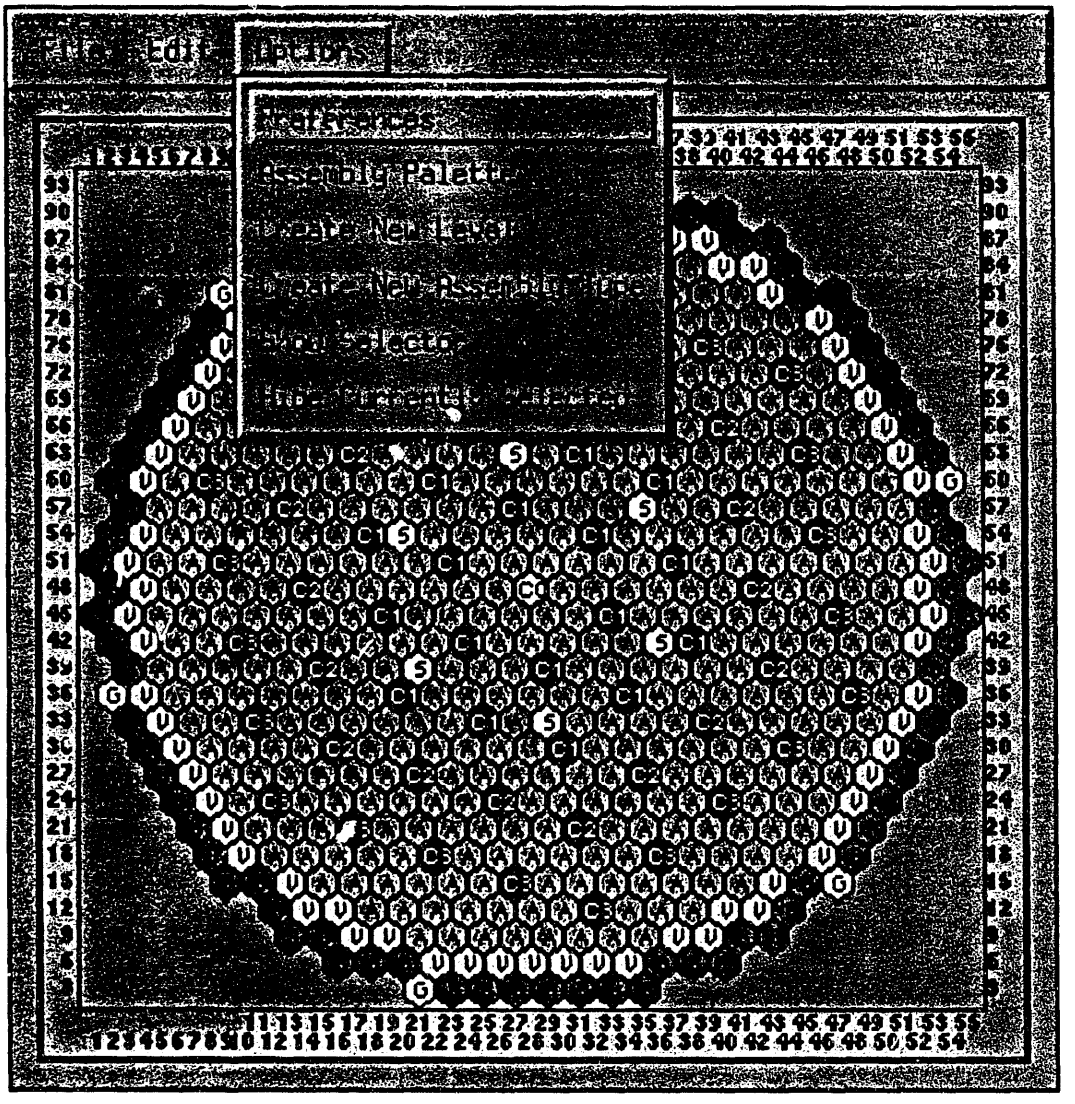


Figure 1. FaceMap Panel for the JOSHUA FM Function

IV.2.2.2 *Assembly Palette Panel*

The Assembly Palette Panel is shown in Figure 2. The Assembly Palette contains a menu of assemblies that already exist, i.e., have already been created and for which complete named data records exist in the read-path hierarchy. The user selects assemblies from this list by clicking (with the mouse) on an assembly name in the Palette. The response to this select procedure depends on the state of the application. If positions in the reactor map were previously selected, then the assembly type from the Palette will be placed in each of those reactor facemap positions. If no facemap positions were selected, then the assembly from the Palette will be loaded into the Assembly Editor.

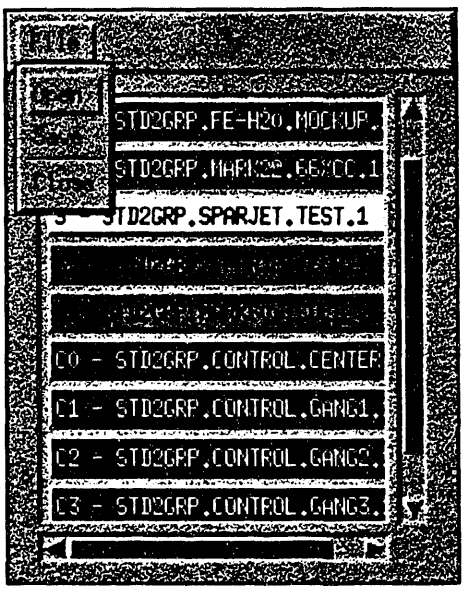


Figure 2. Assembly Palette Panel for JOSHUA FM Function

IV.2.2.3 *Assembly Editor Panel*

The Assembly Editor Panel, shown in Figure 3, is used to define (i.e., create and edit) assemblies.

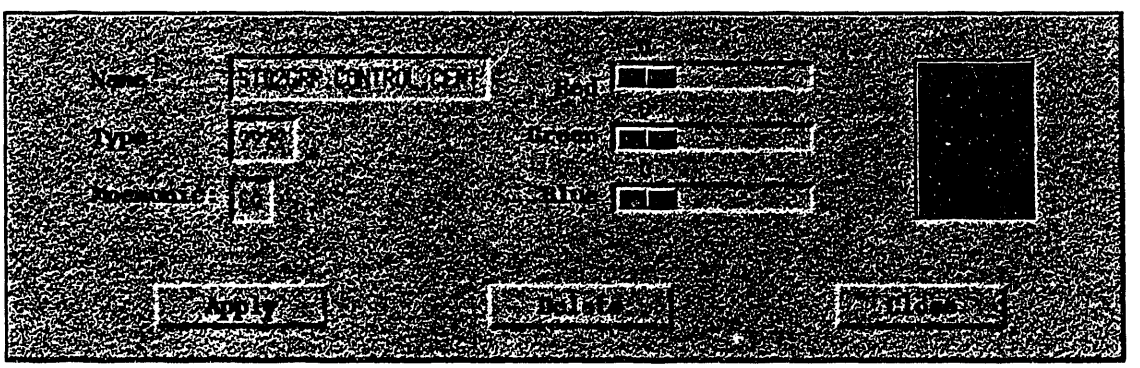


Figure 3. Assembly Editor Panel for JOSHUA FM Function

An assembly definition consists of a name, type, mnemonic and color. The name identifies the desired GLASS record of cell averaged cross section data. The type and mnemonic are used only for compatibility with the current set of codes and make the FM function backwards compatible with existing RM-created records. The color permits the charge designer to graphically differentiate between assembly types. When the user creates an assembly type and presses the apply button, the assembly is placed in the Assembly Palette.

IV.2.2.4 *Preferences Panel*

The Preferences Panel , shown in Figure 4, is used to set the application display to the users choice.

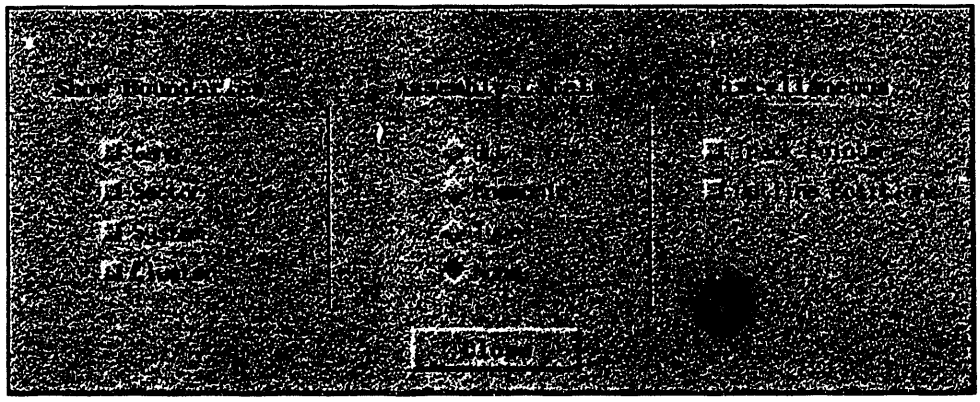


Figure 4. Preferences Panel for JOSHUA FM Function.

Sector, system, gang and cluster boundaries may be toggled and, when 'on', will be displayed on the FaceMap Panel as white lines. The positions of the FaceMap can be displayed with OLC#, mnemonic, type or no label. Pointer tracking, a crosshair that follows the mouse and spans the length and width of the FaceMap Panel, can be toggled. A black outline can be placed on the positions of the FaceMap Panel to help differentiate two similar colored adjacent positions.

IV.2.2.5 *Selector Panel*

The Selector Panel, shown in Figure 5, is used to specify symmetry options and identify specific groupings and type of reactor positions. This panel consists of four small facemaps, each of which represents a specific grouping of reactor positions; namely, gangs, sectors, systems, clusters. The user selects a grouping by pressing the mouse button while the pointer is inside a particular grouping. To copy sector #1 to sector #3, the user 'selects' sector #1 and chooses *copy* from the *Edit* menu on the FaceMap display. Then the user selects sector #3 and chooses *paste* from the same Edit menu. Extended selections can be made by holding down the shift key while pressing the mouse button.

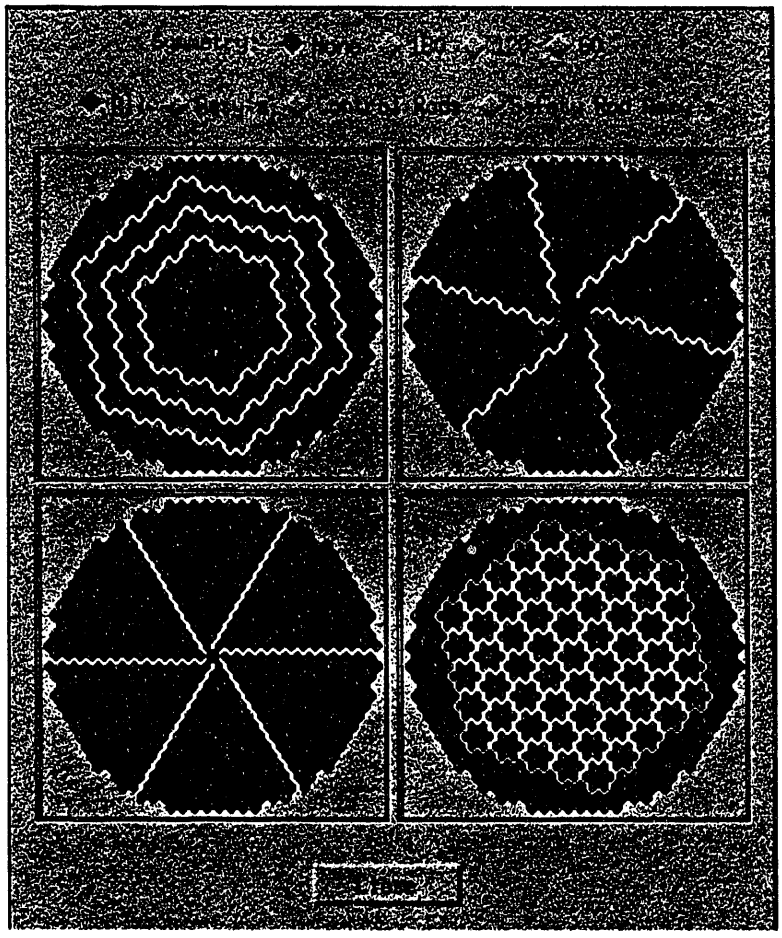


Figure 5. Selector Panel for JOSHUA FM Function

The selector also has two sets of toggle buttons which affect symmetry (e.g., 60^{o}, 120^{o}, 180^{o}, 360^{o}) and position type. The symmetry option changes the way selecting works. With 60^{o} symmetry a selection is replicated in each sector. The position type toggles affect which type of positions (assemblies, control rods, safety rod assemblies) are being referenced with the selector panel. For example, if gang 2 and the *Control Rod* toggle were selected then the positions referenced would be all the control rod positions in gang 2.

V. FUTURE

Consistent with ASCENT's charter, the Team will continue to focus on the following interconnected data management tasks: (a) I/O data management or symbol manipulation, which comprises the interface between the user and the I/O data base; (b) computational data management, which comprises the interface between the numerically intensive part of the computer code and the corresponding computational, or 'internal', data base; and (c) interface, or flow of data, between the I/O data base and the computational data base.

Appendix P:

OUTLINE OF RESEARCH PLAN FOR THE LOS ALAMOS NATIONAL LABORATORY TASP MODEL

Contact: Doug Roberts (505) 667-4569

1. Background

The Los Alamos National Laboratory (LANL) Technical Assessment and Selection Panel (TASP) modeling effort began in the summer of 1990, shortly after the formation of the DOE TASP committee. Prior to starting work at developing a TASP model, LANL conducted a survey of existing models to determine if there were any that could meet the TASP analysis requirements. When it was determined that none of the existing models could sufficiently address the issues attendant to TASP and Complex 21, LANL developed a prototype model specifically designed to address plutonium reconfiguration issues. The initial prototyping effort resulted in a model prototype that was deemed effective in addressing TASP reconfiguration issues. It should be noted that the relatively short prototype development period was possible because of previous, similar modeling work performed at LANL.

Since that time, the TASP model has been enhanced on an as-needed basis to address new analysis requirements as they arose. The original prototype was designed to address plutonium issues only, and so a significant amount of effort has been spent making the model more generic so that uranium, lithium, tritium, and HE material processing can be more easily modeled. Also, additional modeling requirements have been identified as the user base has expanded.

In parallel with the support efforts for the LISP-based version of the TASP model, LANL is investigating alternative software and hardware environments for future implementations of the model. The intent is to develop a new version of the TASP model that will be faster and easier to use. Recently, several new software development environments have become available which appear to have several advantages over the current model software environment. These advantages include

- increased speed
- increased portability across hardware platforms
- greater ease of use
- richer development environment
- less cost.

2. Model Description

2.1. General Description

The TASP model is an object-oriented discrete-event / continuous simulation system designed to address TASP reconfiguration issues. The current implementation was developed using KEE[13]

[13] KEE (Knowledge Engineering Environment) is a trademark of IntelliCorp of Mountain View, California.

and LISP software. The model currently is running on Sun Microsystems[14] SPARC workstations. The nature of the TASP Complex 21 project requires that the analysts evaluate different technologies and their impact on production, waste generation, and facility utilization. In order to supply the analysts with this information, the model is sufficiently detailed that the impact of one technology over another can be measured.

To evaluate the overall impact of technology comparisons, it was necessary to directly model the technologies. Therefore, the TASP model is at the level of detail of individual processing operations, or processes. Examples of these processes are machining, casting furnaces, anion exchange columns, etc. To assist the analyst in this task, the model was designed to allow flowcharts to be quickly input and modified.

Within the scope of the Complex 21 study, it is necessary to predict primary material flow rates to determine if production schedules can be met. It is also necessary to predict the flow rates of secondary and tertiary residue and waste streams since waste generation will be an important driver in the design of Complex 21. For this reason, a mass balance approach to system modeling was adopted by the TASP model. A full mass balance characterization of each process being evaluated allows the analyst to accurately predict overall facility behavior, and to measure the full impact of technology comparisons in terms of residue and waste generation, as well as in terms of product throughput.

To accurately predict residue and waste stream generation, the analysts using the TASP model have identified several specific types of waste-generating behavior that the model was required to emulate. These fall into two categories: periodic waste generation, and process waste generation. An example of periodic waste generation is the gloves and paper wipes produced from periodic glove box maintenance, wipe-down, and material inventory. In some cases this occurs every 30 days. An example of process waste generation is when acid is added to the system during aqueous residue recovery operations. Each liter of acid added to the system results in a defined number of cubic feet of cement required in waste disposal operations. Both types of waste generation are emulated in the TASP model.

After all of the technologies have been identified that will comprise a given portion of Complex 21, the facilities will need to be sized and load-leveled to meet the build and retirements schedules. The TASP model can assist the analyst when sizing facilities.

One method of sizing makes use of the utilization factor for a process. The utilization factor is defined as the percentage of the available processing time in a given period that the process was busy. If, after a simulation sizing run, the analyst observes that one process had a high utilization factor (perhaps 90%) and the processes downstream for it all have low utilization factors, then a probable bottleneck in the system has been identified. The analyst can then add processing capacity to the process in question (perhaps by adding another piece of equipment in parallel) and re-run the simulation. This iterative use of the model is called "load levelling."

A similar measure used in sizing a facility is to measure the queue of material upstream of each process. Those processes with large queues will be suspected bottlenecks.

2.2. Current State of Implementation (March 1992)

The TASP model is presently in use by analysts at Los Alamos, Livermore National Laboratory, and at Martin Marietta Energy Systems at the Y-12 Plant in Oak Ridge. In addition, analysts from

---

[14] Sun Microsystems is a registered trademark of Sun Microsystems, Inc. of Mountain View, California.

the Savannah River Plant have been using the model at Los Alamos until they have their own internal capability. We are presently in discussion with personnel at the Pantex Facility in Amarillo, Texas, for potential model application there.

To date, the model has been primarily used for plutonium analysis. In the last three months, however, Y-12 personnel have initiated lithium and uranium analysis efforts. Preliminary verification and validation efforts have been completed, with the assistance of Livermore personnel. The formal documentation and verification and validation tasks are planned to begin at Los Alamos soon.

3. Current Status

3.1. Model Status

The TASP model has been developed to the stage where it can address most (of the now known) plutonium analysis issues in site return, purification, and manufacturing technologies. Since aqueous plutonium processing has not yet been completely modeled, there may be new requirements places on the model in order to complete aqueous plutonium modeling. The model is starting to be used at Y-12 to address lithium and uranium issues, and several new features have been identified that must be incorporated into the model before it can be used to perform a complete set of lithium or uranium reconfiguration runs. A prioritized list of additional features for the model is being maintained and implemented at Los Alamos.

3.2. Analysis Status

The data set supporting the Design Guidance Manual for the proposed future Plutonium facility has been delivered to Fluor Daniel Corporation (the prime contractor for the reconfiguration effort). Efforts are continuing to fill remaining holes in the data for the Complex-21 Plutonium model. Because the Plutonium flowsheet is still changing, progress toward a complete data set has been slower than expected. Nonetheless, the model is currently being used to determine material flows throughout the Plutonium flowsheet and to assess the impact of reagent recycle on the Nitrate Aqueous Recovery module. Progress in the Uranium data set and flowsheet is proceeding rapidly, with two new users now being trained in the use of the TASP model. Progress in the Salt Flowsheet and data set has reached the stage where the TASP model can be used by Y-12 analysts to study technology trades proposed by the TASP Salt Working Team.

A second generation database manager has been created with Apple's HyperCard utility to both manage TASP-Complex-21 data and to automatically create TASP model input files from that data. At present, the most recent data for Site-Returns, Manufacturing, Chloride, and Nitrate operations in the Plutonium model have been entered into the database. The database has successfully created valid Baseline, Scenario, and Disassembly-Schedule input files for Plutonium Chloride operations; input files for other section are under development. Future plans include adding the ability to read data from the existing Macintosh/IBM-PC Excel databases, the ability to read data from existing input files, and the automatic generation of module flow diagrams.

A modelling workshop was held in mid-March to detail progress in the TASP modelling effort. TASP model users from Lawrence Livermore, Los Alamos, Savannah River Site, and Y-12 provided presentations. Fluor Daniel also provided a presentation on the architect/engineer perspective. Future actions were defined, focusing upon documentation needs for a user manual and a quality assurance plan.

3.3. Quality Assurance Program

A quality assurance plan is being developed for the TASP model to meet the requirements of Los Alamos software development standards. Documentation will be supplied on code specifications, design description, and verification and validation of code results. Once developed in a draft form, these documents will be circulated to user groups for comment. User comments are needed to ensure consistency of Los Alamos QA requirements with those of user sites.

Appendix Q:

ROCKY FLATS PLANT SIMULATOR

EG&G Rocky Flats

Contacts: Cheryl Steinmeyer (303) 966-7407
Terry Hill (303) 966-4065
FAX: (303) 966-2241

Description:

The Rocky Flats Plant Simulator (RFPS) is an object oriented, discrete event, Artificial Intelligence (AI) software system used to model activities supportive of the Rocky Flats Plant mission. Originally started as the Defense Program Simulation/Rocky Flats Plant (DPS/RFP) project, the system will be utilized to answer high-level, what-if questions concerning residue elimination, shipping, liquid waste treatment, waste management, nuclear material storage, and plutonium discard limit scenarios. Models are developed in a SUN/UNIX workstation environment, using Common List Processing (Common LISP) and Knowledge Engineering Environment (KEE) programming tools.

The project began in 1988 with emphasis placed on modeling the manufacturing, aqueous recovery and pyrochemical processing buildings. With the recent change in the Rocky Flats mission, the modeling efforts have been redirected to include residue elimination, stabilization of liquid stored in bottles and tanks, nuclear material consolidation, liquid waste treatment, waste management, shipping, and laboratory operations. Current development is on the liquid stabilization effort as well as improvements to the core of the model and the pre and post processors.

The detail of the simulation includes individual pieces of equipment, equipment downtime, operating personnel, certifications, reagent usage, residue and waste generation, storage, plutonium discard limits, and detailed process decisions. RFPS will assist in obtaining an in-depth understanding of the capabilities and capacities available, identify and resolve system limitations and bottlenecks and develop the consequences of undesired or projected events.

Since the change in the plant's mission, the need for RFPS has increased greatly. RFPS will assist in answering questions such as "What if we do not start up this building?"; "What if we use process x to eliminate residues?"; What if we move all of the nuclear material out of one building and store it in another?"; What if we must operate with x number of operators?"; and "What if we cannot ship any waste for x years?".

For further information, contact:
Cheryl Steinmeyer, Terry Hill
EG&G Rocky Flats
P. O. Box 0464
Building 371
Golden, Colorado 80402-0464

Appendix R:

AUTOMATICALLY PROGRAMMED METROLOGY

Martin Marietta Energy Systems Inc., Oak Ridge Y-12 Plant

Contacts:	Claude Begley	(615) 574-3221
	Ed Klages	(615) 574-1869
	Rob Wilson	(615) 576-3678
	FAX:	(615) 574-5458

Description

An inspection system, Automated Programmed Metrology (APM), which automates and integrates the planning and program creation activities for inspection on coordinate measurement machines (CMM) has been designed and is currently being productionized at the Oak Ridge Y-12 Plant. APM integrates electronic product definition, inspection planning and CMM programming. Each of these tasks, implemented in the Model Enhancement Module, Inspection Plan Generator, and Program Generation Module rely on a integrated object oriented data structure to accomplish their tasks. Modern object oriented programming methods using the C++ language are being used to implement the APM system. User interface is provided through the CAD system interface, where the base geometric and tolerance information is gleaned, and through a set of Motif windows which present inspection plan and program information in a format which is both easy to read and to manipulate. APM will be integrated with the plant's design and manufacturing electronic file system, thus tieing it into the business stream.

The Model Enhancement Module of APM allows part files from Y-12's manufacturing CAD system to be supplemented with additional information needed to support inspection activities. A design engineer, using menus integrated with the CAD system, specifies information such as relationships between tolerance symbols and part geometry. The resulting model, a combination of the original CAD data and the supplemental data, is represented in an object oriented structure which integrates all part aspects needed to support inspection. The intent of the Model Enhancement Module's design is that it be easily adaptable to most traditional CAD or solid modeler systems. This will insulate the core inspection functionality from the implementation modeler. APM thus does not require that the modeler be in compliance with a complex representation, such as PDES, but rather that it provide a basic set of interface routines thus allowing use of the information which it can provide.

The Inspection Plan Generator of APM allows a dimensional inspection engineer to create a high level inspection plan. The inspection engineer enters a variety of administrative information and selects tolerances to included in the plan. Since the Inspection Plan Generator is integrated with the object database created by the Model Enhancement Module, the system can automatically access tolerance and geometric information, apply inspection rules contained in its knowledge base and then present a recommendation for an inspection strategy. The inspection engineer can query the system as to the logic used in arriving at a strategy and if desired override the strategy. This combination of system and inspection engineer input allows for a largely automated but still flexible system.

The Program Generation Module of APM allows a CMM programmer to generate a program plan, which is a detailed version of the inspection plan, and subsequently generate an inspection program. When creating the program plan the CMM programmer enters additional administrative

information, determines the inspection probe configuration, and specifies parameters such as search and retract distance. The CMM programmer adds detail regarding touchup and intermediate probings required to support the measurements comprising the inspection plan. As in the Inspection Plan Generator, the Program Generation Module uses a rule base to suggest programming strategies, preferred coordinate systems, ordering of measurements, parameter values, and methods for making intermediate probings. Upon completion of the program plan the CMM programmer can request that a collision-free DMIS program be automatically generated - ready for simulation or postprocessing to a specific CMM for execution.

In summary, APM is production oriented inspection system designed to integrate inspection activities from planning to program generation. It makes use of current object oriented technology and windowing interface technology on UNIX workstations. It has been designed to provide the right mix of automation, flexibility and adaptability.

For more information, contact:

Claude Begley
Martin Marietta Energy Systems, Inc.
P.O. Box 2009
Oak Ridge, TN 37831-8125

Appendix S:

Electronic Notarized Document System (AMENDS)

Martin Marietta Energy Systems, Oak Ridge

Contact: Mary Theofanos, fft@msr.epm.ornl.gov, (615) 576-6660

Description:

This project designed and developed a prototype electronic notarized document system to address the needs of the "paperless office". The prototype involved two major efforts. The first effort required research and development of a mathematically secure electronic signature capability based on national encryption standards to provide unforgeable and authenticatable legal signatures. The second, focused on the design and development of a records mangement system to provide an on-line document storage and retrieval system incorporating the electronic signature capability to protect against document falsification.

The electronic signature facility prototyped in AMENDS provides for 3 levels of authentication. The first level of signature authentication is designed to emulate the process of an individual signing a document. This signature uses an arbitrated signature scheme, incorporating a message digest algorithm (referred to as MD4) applied to the document, applying DES as the encryption technique to the digest, all using a secure communication protocol based on the Kerberos approach. The level 2 authentication provides a notary capability, similar to the notary in the paper world. Finally, the third level address the problem of alterable documents by incorporating an archiver and introducing write-once-read-many (WORM) optical disk storage technology.

The projects first phase developed a proof-of-concept prototype using open systems standards and a client-server architecture running on a POSIX development platform (Sun workstation) demonstrating the electronic signature capability and a baseline document management system. The second phase enhanced the prototype by adding the following features to the records management system:

- a graphical user interface
- a means of creating, managing and querying electronic documents
- a means of creating and editing document forms
- a means of creating and editing document routings
- a means of document query by word combinations
- a means of monitoring and managing long-term storage requirements.

The project included research into the use of object-oriented technologies for the design and development of the system in order to evaluate the value of object-oriented technology. Thus the software implementation language is C++ and the database managment system is an object-oriented DBMS (CDM) demonstrating the use of object-oriented technology. Several OODBMSs were examined and tested before selecting CDM. A comparison of several of their individual features is presented in the System Design Documentation. In addition, considerable metrics have been developed and collected showing the value of the object-oriented technology in this application. For instance, the average number of lines of code generated per labor month was 686.

Currently, AMENDS addresses the needs of the paperless office as envisioned by MICOM and provides document coordination and version mangement including auditing capabilities as well as security through electronic signatures. Thus AMENDS can be the primary integration tool for data

resource management. In addition, the base system has been ported to MICOM Unisys 5000 UNIX equipment and distributed for a networked environment.

Several enhancements have been proposed for future phases. These would extend the capabilities in order for AMENDS to become a complete integration tool for information resource management.

Deliverables for the project include:
- Electronic Signatures Document, K/DSRD-472, June 1990
- Prototype Electronic Records Management Software Design Document, K/DSRD--471, July 1990
- Proof-of-Concept Prototype, September 1990
- Final System Design Document
- User Documentation

A Sun workstation prototype system is available for demonstrations.

Project Sponsor: Department of the Army, US Army Information Systems Command - MICOM

Project began: March 1990, scheduled for completion Spring 1992

Principal Investigator: Mary F. Theofanos, Data Systems Research and Development (DSRD)

Address: Martin Marietta Energy Systems
P. O Box 2003
Oak Ridge, TN. 37831--7346

FAX: (615) 574-9955

Appendix T:

OBJECT ORIENTED DEVELOPMENT OF AN EXPERT SYSTEM FOR PRIORITIZING[15]

P. Craig Hopson
Westinghouse Savannah River Company[16]
Savannah River Site, Aiken, SC 29808

Abstract

Prioritizing list of diverse entities such as projects, tasks, documents, recommendations or physical locations is a necessary part of business at DOE facilities. A key issue is whether or not this necessary but often problematic activity of prioritizing is performed in a methodical, defensible and traceable manner. Sound methods of prioritizing are often not employed because of their complexity or difficulty in implementation. To overcome these problems, WSRC is developing an expert system, First Priority, which will provide individuals or committees a comprehensive process for prioritizing lists of any sort in the difficult case where there are several goals which are hard to compare and measure. A set of windows, editors, and pull-down menus guide the user in building and modifying an (inverted) weighted tree structure which represents the goals the prioritization is to advance. The process has four stages which are generally followed in order. They are

- building the goal tree,
- ordering the goal tree nodes,
- weighting the goal tree nodes, and
- designing measurement methods for each leaf node

Based on the resultant structure an evaluation module is generated to evaluate the items of the list. This list is then prioritized and grouped into user-defined categories, taking into account cost or other resources. Additional First Priority tools provide sensitivity analysis, graphical display of data, and reporting.

Implementation

First Priority is being developed for the Apple Macintosh using MPW C++from Apple Computer and Apple's object oriented application framework call MacApp. This decision was made for several reasons. First, nearly everyone who would have a need to use First Priority at SRS has a Macintosh on his desk. Because the Macintosh user interface is consistent from application to application , the amount of time required to become productive with the program will be minimized; limited to learning the methodology. Targeting the Macintosh also allows us to take advantage of Apple's leading edge object-oriented technology including MacApp, an extensive class library written in C++ which implements the Macintosh interface, and other assorted development tools such as view and resource editors and class browsers.

---

15 Savannah River Site document number WSRC-RP-92-480

16 This contribution was prepared in connection with work done under Contract No. DE-AC09-89-SR18035 with the U.S. Department of Energy.

Once the decision was made to pursue an object-oriented implementation of First Priority, it became apparent that a new design methodology would have to accompany that paradigm shift; available methodologies simply don't work for OOD. We are investigating a new methodology for software development called Solution Based Modeling (SBM). SBM is a complete methodology for developing object-oriented software from requirements analysis through product development. Its notational system, the visual Design Language (VDL) allows analysis, design, and programming concepts to be expressed graphically and in a manner that allows formal verification of the requirements.

Appendix U:

Y-12 Capabilities System

Martin Marietta Energy Systems, Oak Ridge

Primary Contact: C. Ray Riggs, Y-12 Program Management, riggscr@ornl, (615) 574-5814
Computing Contacts: Betty Lou Alspaugh, alspaughbl@ornl, (615) 574-9235
Deanna Barnett, bqq@ornl, (615) 576-4202

The Y-12 Capabilities System is a multimedia, multi-purpose information system being developed to support technology transfer efforts. The Y-12 Capabilities System currently shows users a wide range of the Y-12 Plant's diversified capabilities, from a broad overview of the plant to such detailed specifications as machine accuracy, machine features, and where to get additional information. Text and high resolution color photographs are currently being used to depict the plant's capabilities. An important project requirement is portability, that is, delivery of the system on a notebook-sized personal computer capable of showing 256 colors at 640x480 resolution. Future plans are to add animation and/or video capabilities along with sound to enhance the system and to populate the dBASE databases which feed the system to cover more Energy Systems capabilities.

Depending on the user's interest, the Capabilities system can lead the user through a hierarchy of information. This hierarchy starts with an overview of the plant's capabilities, e.g. from the concept and design phases, through manufacturing capabilities and QA, to technology transfer. Also included at this level is a list of unique capabilities, that is, a list of activities, systems, programs, and applications which are considered some of the specialties of the Y-12 Plant. Much of this hierarchy of information is maintained in dBASE database files.

The Y-12 Capabilities system was written using Information Builder's Level5 Object product to run on an IBM or compatible PC under the windows environment (Windows 3.0 or higher). Level5 Object is an object-based expert system shell that provides an interactive windows-based user interface. Level5 Object also offers an interface with relational data base models (dBase and FOCUS products), hypertext capabilities, graphical development, and debugging tools.

The ease of developing the user interface was one of Level5 Object's strongest features. Not only is the user interface easy to develop, it's also fairly easy for users to use. Windows-type pushbuttons are used to allow the user to progress through the information hierarchy. Very little documentation is needed to run an application. Deriving the data from a database, rather than from internal classes, was also fairly easy, thought it took several tries to make Level5 utilize the database's index file for quick retrieval of data. Level5 Object does have some problems utilizing the inheritance feature from class to class and this is definitely a indrance. Hopefully the next version of the product will correct the inheritance problem and will make Level5 Object a true object-oriented system.

Milestones: Project Initiation: Aug. 1991
Prototype completed: March 1, 1992
First phase of the production system completed: April 24, 1992

Programmers' Address:
Martin Marietta Energy Systems
P. O. Box 2009
Oak Ridge, TN 37831-8227

Appendix V:

KATIE

Martin Marietta Energy Systems - Oak Ridge

Contact: Abigail G. Roberts
Martin Marietta Energy Systems, Inc.
Y-12 Plant, P. O. Box 2009, MS-8066
Oak Ridge, TN 37831-8066
Phone: 615: 574-5701
Fax: 615: 574-0334

Knowledge-based Assistant for Troubleshooting Industrial Equipment (KATIE)

The primary feature of KATIE is the interactive delivery of technical information to the end-user. The system's knowledge base consists of step-by-step maintenance, repair and calibration procedures; video images of each procedure step; the text of maintenance manuals with links to procedure steps; and an expert system front-end which performs initial problem isolation.

KATIE uses an IBM PC/AT with video capture, compression, and display board; video mixer board; VGA-type monitor; and 380 Mbyte ESDI hard drive with 8 Mbyte memory. KATIE was developed using DOS, Smalltalk V/286, and C.

Project completion date was September 1990.

Knowledge-based Assistant for Training and Information Exchange (KATIE, version 2)

This system is being developed for the Y-12 Security Division as an interactive training program for DOE certification and recertification of security operators. The system uses the original KATIE format with additional audio and video features.

KATIE is a multimedia application using a 486 33MHz PC with a VGA monitor and 16 Mbyte of memory; a video capture, compression and display board; an audio processing board; and a dual 90 Mbyte Bernoulli drive.

The current status on this project is entry of Y-12 Security training information and Security specific software enhancements.

Appendix W:

Trim-Sol Multimedia Training System

Martin Marietta Energy Systems - Oak Ridge

Contacts:

Larry Hopper	615: 576-5271	Computing & Telecommunications Division
J. T. Greer	615: 574-1317	Y-12 Development Division

Description:

Starting in March 1991, the Trim-Sol multimedia training system was developed to help train individuals at the Y-12 Plant in the proper procedures for collecting, purifiying, and redistributing Trim-Sol machine coolant.

The system was implemented using the Guide Product from OWL on a IBM-compatible personal computer and Windows 3.0. The first phase of the system used hypertext and "active" pictures to present information and procedural assistance. The system has several windows: one for the text of the formal work procedure, several mouseable buttons for navigating through the document, one for graphics and video, and another window for definitions of works used in the procedure. By clicking on navigation buttons or highlighted words or phrases, the user can learn about the procedure or be assisted in performing it.

Use of a computer-controlled video cassette recorder was evaluated during the summer, and it was determined that addition of video would take several months to develop. The system could be beneficial without video, so it was delivered to the Fabrication Division for use late in 1991. Addition of video segments is still a possibility for future development, but the need is reduced due to the changing mission of the Y-12 Plant.

Appendix X:

OBJECT ORIENTED TRAINING AT THE SAVANNAH RIVER SITE[17]

E. L. Funderburk
Westinghouse Savannah River Company[18]
Savannah River Site, Aiken, SC 29808

Description:

Training is an essential element of the effort to introduce the implement Object Oriented technology and its benefits at the Savannah River Site. The Scientific Computing Resource Center (SCRC), operated within the Savannah River Technology Center as focal point for scientific and technical computer training, currently offer a limited selection of courses aimed at computer professionals who wish to learn about the Object Oriented paradigm and how to apply it to the entire range of their development activities, from analysis and design to coding and implementation.

The SCRC operates on a zero based budget, and must fund its activities through direct charges to the student. This approach has proven to be cost-effective, in that the typical course cost to the student is roughly equivalent to what the tuition alone would have cost for an off-site offering. The travel and living expenses associated with an off-site course are avoided, as is the time away from the job to travel to and from cities in which such courses are offered.

The following describes the current course offerings, as well as key courses currently being planned for the near future.

Introduction to the Object Oriented Paradigm

Description: This is a one-day, language-independent, comprehensive seminar for technical managers, analysts, software designers, and programmers whos wish to understand the fundamental concepts and advantages of the Object Oriented paradigm.

Content:
Upon completion of the course, the student will:

- Understand the advantages that the Object Oriented paradigm brings to the issues of:
  - code reuse
  - portability
  - code maintenance
  - the development of very large systems.
- Understand the software life cycle using the object oriented paradigm.
- Become familiar with graphical notation as used in the object model.
- Trace the software development process from analysis through design and implementation
- Have an understanding of the basic concepts of object oriented programming such as data abstraction, encapsulation, inheritance, and polymorphism.

---

17 Savannah River Site document number WSRC-RP-92-481

18 This contribution was prepared in connection with work done under Contract No. DE-AC09-89-SR18035 with the U.S. Department of Energy.

Object Oriented Analysis and Design

Description: This course is designed for students who have completed the suite of C++ programming courses offered on -site, or completed a medium to large software project in some object oriented language. At lease a reading knowledge of C++ is required.

Content:

- Students will successfully analyze a sample problem demonstrating their understanding of such object oriented design issues as:
  - abstraction
  - generalization versus specialization
  - composition
  - state and object models.
- Based on the above analysis, students will produce a conceptual design which includes the appropriate:
  - classes
  - subsystems
  - interactions
- Followed by a detailed design which will include
  - inheritance
  - composition
  - delegation
  - state transition.

C++ for Non-C Programmers

Description: This is an introductory course for technical programmers and covers the basic concepts of C++ and Object Oriented programming. Programming experience in a high level language (e.g. Pascal, Fortran, etc.) is required.

Content:
Upon completion of the course, the student will:

- Have a basic understanding of the following object oriented concepts:
  - abstraction
  - classes and objects
  - inheritance
  - function and operator overloading
  - dynamic binding.
- Have working knowledge of C++ built-in data types.
- Have the ability to implement arrays and pointers.
- Understand the use of functions and control statements in C++.
- Understand dynamic memory allocation.
- Have the ability to compile, link, test, and debug simple programs.

Object Oriented Programming in C++

Description: This course in intended for users who wish to learn the object oriented paradigm by applying the concepts in application development using C++. Previous experience with C++ or a structured programming language is required.

Content:

Topics include:
- object oriented programming
- benefits of object oriented methods
- C++ syntax
- the structure of a C++ program
- C++ constructs
- C++ types
- advanced C++ classes
- user defined classes
- public and private objects
- reusable code libraries
- object oriented design techniques
- comparison of object oriented languages
- dynamic memory allocation
- recursion

Intermediate C++

Description: A course for students with a working knowledge of C or C++. Six months programming experience in C or C++ or successful completion of "C++ for Non-C Programmers" is required.

Content:

Upon completion of the course, the student will have completed labs demonstrating an understanding of:
- the concepts of data abstraction and encapsulation
- initialization and cleanup (constructors and destructors) instance variables
- single inheritance
- function and operator overloading
- dynamic binding
- exception handling.

In addition to lab exercise the student will be introduced to:
- parametric types
- the concepts of multiple inheritance.

DATE
FILMED
3/11/93