X/Open Technical Study

Getting into Objects

X/Open Company Ltd.

X/Open Technical Study

Getting into Objects

ISBN: 1-85912-032-6
X/Open Document Number: E402

Published by X/Open Company Ltd., U.K.

Any comments relating to the material contained in this document may be submitted to X/Open at:

X/Open Company Limited
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

XoSpecs@xopen.co.uk

Contents

Preface

X/Open

X/Open is an independent, worldwide, open systems organisation supported by most of the world's largest information systems suppliers, user organisations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems.

X/Open's strategy for achieving this goal is to combine existing and emerging standards into a comprehensive, integrated, high-value and usable open system environment, called the Common Applications Environment (CAE). This environment covers the standards, above the hardware level, that are needed to support open systems. It provides for portability and interoperability of applications, and so protects investment in existing software while enabling additions and enhancements. It also allows users to move between systems with a minimum of retraining.

X/Open defines this CAE in a set of specifications which include an evolving portfolio of application programming interfaces (APIs) which significantly enhance portability of application programs at the source code level, along with definitions of and references to protocols and protocol profiles which significantly enhance the interoperability of applications and systems.

The X/Open CAE is implemented in real products and recognised by a distinctive trade mark — the X/Open brand — that is licensed by X/Open and may be used on products which have demonstrated their conformance.

X/Open Technical Publications

X/Open publishes a wide range of technical literature, the main part of which is focussed on specification development, but which also includes Guides, Snapshots, Technical Studies, Branding/Testing documents, industry surveys, and business titles.

There are two types of X/Open specification:

- *CAE Specifications*

  CAE (Common Applications Environment) specifications are the stable specifications that form the basis for X/Open-branded products. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Developers who base their products on a current CAE specification can be sure that either the current specification or an upwards-compatible version of it will be referenced by a future X/Open brand (if not referenced already), and that a variety of compatible, X/Open-branded systems capable of hosting their products will be available, either immediately or in the near future.

  CAE specifications are published as soon as they are developed, not published to coincide with the launch of a particular X/Open brand. By making its specifications available in this way, X/Open makes it possible for conformant products to be developed as soon as is practicable, so enhancing the value of the X/Open brand as a procurement aid to users.

- *Preliminary Specifications*

  These specifications, which often address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations, are released in a controlled manner for the purpose of validation through implementation of products. A Preliminary specification is not a draft specification. In fact, it is as stable as X/Open can make it, and on publication has gone through the same rigorous X/Open development and review procedures as a CAE specification.

  Preliminary specifications are analogous to the *trial-use* standards issued by formal standards organisations, and product development teams are encouraged to develop products on the basis of them. However, because of the nature of the technology that a Preliminary specification is addressing, it may be untried in multiple independent implementations, and may therefore change before being published as a CAE specification. There is always the intent to progress to a corresponding CAE specification, but the ability to do so depends on consensus among X/Open members. In all cases, any resulting CAE specification is made as upwards-compatible as possible. However, complete upwards-compatibility from the Preliminary to the CAE specification cannot be guaranteed.

In addition, X/Open publishes:

- *Guides*

  These provide information that X/Open believes is useful in the evaluation, procurement, development or management of open systems, particularly those that are X/Open-compliant. X/Open Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming X/Open conformance.

- *Technical Studies*

  X/Open Technical Studies present results of analyses performed by X/Open on subjects of interest in areas relevant to X/Open's Technical Programme. They are intended to communicate the findings to the outside world and, where appropriate, stimulate discussion and actions by other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism for X/Open to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of an X/Open technical activity. Although at the time of its publication, there may be an intention to progress the activity towards publication of a Specification, Guide or Technical Study, X/Open is a consensus organisation, and makes no commitment regarding future development and further publication. Similarly, a Snapshot does not represent any commitment by X/Open members to develop any specific products.

Corrigenda

Most X/Open publications deal with technology at the leading edge of open systems development. Feedback from implementation experience gained from using these publications occasionally uncovers errors or inconsistencies. Significant errors or recommended solutions to reported problems are communicated by means of Corrigenda.

The reader of this document is advised to check periodically if any Corrigenda apply to this publication. This may be done either by email to the X/Open info-server or by checking the Corrigenda list in the latest X/Open Publications Price List.

To request Corrigenda information by email, send a message to info-server@xopen.co.uk containing the line:

```
request Corrigenda; topic index
```

This Document

This document is a Technical Study (see above). It considers the relevance of object-oriented technologies to open systems and the possible technical strategies that X/Open may adopt for the specification of a complete object-oriented environment for distributed applications.

The evolution of object-oriented technology continues apace. It is probable that some of this document's statements as to the current picture and likely developments will be overtaken by events.

Structure

This study is divided into three chapters:

- Chapter 1 gives an overview of the key features of object-oriented technologies and the benefits which, it is claimed, arise from their use. This chapter also describes the specifications being developed by the Object Management Group.
- Chapter 2 draws together various strands of opinion about the current and future exploitation of object-oriented technologies in the open systems arena. The bulk of the information underlying this section was obtained from a series of interviews conducted with representatives of a number of X/Open member companies.
- Chapter 3 gives a broad indication of X/Open's future directions on OOT and the way in which a cohesive strategy will evolve.
- A glossary and index are provided.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for options to commands, filenames, keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  - — command operands, command option-arguments or variable names, for example, substitutable argument prototypes
  - — environment variables, which are also shown in capitals
  - — utility names
  - — external variables, such as *errno*
  - — functions; these are shown as follows: *name*( ); names without parentheses are C external variables, C function family names, utility names, command operands or command option-arguments.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font.

Trade Marks

X/Open™ and the ''X'' device are trade marks of X/Open Company Ltd.

This list represents, as far as possible, those products that are trade marked. X/Open acknowledges that there may be other products that might be covered by trade mark protection and advises the reader to verify them independently.

Referenced Documents

The following X/Open documents are referenced in this study:

CORBA
: X/Open Preliminary Specification, February 1993, The Common Object Request Broker: Architecture and Specification (ISBN: 1-872630-90-1, P210).

XMOG
: X/Open Guide, September 1993, Systems Management: Managed Object Guide (XMOG) (ISBN: 1-85912-06-7, G302).

XRM
: X/Open Guide, August 1993, Systems Management: Reference Model (ISBN: 1-85912-05-9, G207).

The following non-X/Open documents are referenced in this study:

ARM
: The Annotated C++ Reference Manual, Margaret Ellis and Bjarne Stroustrup, Addison Wesley, 1990.

C++ Library
: Developing the Standard C++ Library, P.J. Plauger, The C Users Journal, October 1993.

C++ Standardisation
: C++ Standardisation: Top Issues, Dmitry Lenkov, Journal of Object-Oriented Programming, June 1992.

Consultants' Conspectus
: Consultants' Conspectus, April 1993, Prime Marketing Publications, 1993.

Distributed Computing Systems
: Object-Orientation in Heterogeneous Distributed Computing Systems, J.R. Nicol, C.T. Wilkes and F.A. Manola, IEEE Computer Volume 26, Issue 6, June 1993.

Dr. Dobbs
: Dr. Dobbs Journal, Volume 18, Issue 11, October 1993.

Essence of Objects
: Essence of Objects: Concepts and Terms, Alan Snyder, IEEE Software Volume 10, Issue 1, January 1993.

Guessing Games
: Programming Language Guessing Games, P.J. Plauger, Dr. Dobbs Journal, Volume 11, Issue 18, October 1993.

Making Objects Persistent
: Making Objects Persistent, Mary E.S. Loomis, Journal of Object-Oriented Programming, Volume 6, Issue 6, October 1993.

Object Database Standard
: The Object Database Standard ODMG-93, R. Cattell ed., Morgan Kauffman, 1993.

Object-Oriented Design
: Object-Oriented Design with Applications, Grady Booch, Benjamin/Cummings, 1991.

Object-Oriented Programming
Object-Oriented Programming an Evolutionary Approach, Second Edition, Brad J. Cox and Andrew J. Novobilski, Addison Wesley, 1991.

Object-Oriented Software Construction
Object-Oriented Software Construction, Bertrand Meyer, Prentice Hall, 1988.

OMA
Object Management Architecture Guide, Object Management Group, Document number 92.11.1, Revision 2.0, Second Edition, 1992.

OSA
Object Services Architecture, Object Management Group, Document number 92.8.4, Revision 6.0, 28 Aug 1992.

Recent Language Extensions
Recent Language Extensions to C++, Dan Saks, The C Users Journal, June 1993.

Stroustrup 1st Edition
The C++ Programming Language, Bjarne Stroustrup, Addison Wesley, 1987.

Stroustrup 2nd Edition
The C++ Programming Language Second Edition, Bjarne Stroustrup, Addison Wesley, 1991.

Taligent White Paper
Lessons Learned from Early Adopters of Object Technology, Taligent White Paper, 1992.

Virtual Interview
Virtual Interview, Mike Vilot, Doug Schmidt, C++ Report, Volume 5, Issue 8, October 1993.

Workshop Report
Workshop on Application Architectures Workshop Report, National Institute of Standards and Technology, 1993.

Chapter 1

Object-Oriented Technology: The Landscape

This chapter presents an overview of object-oriented technologies (OOT) and the major groups which are active in the development of these technologies.

The overall purpose of this study is to explore the relationship between the open systems arena and object-oriented technologies. The two phrases *open systems* and *object-oriented* both seem to be used within the Information Technology community with much the same mixture of truth, hope and blind enthusiasm that is employed when *new* and *improved* are applied to washing powder. Although it may be assumed that the phrase *open systems* has some common meaning amongst the audience of this study, some shared understanding of the concepts underlying the phrase *object-oriented* will be needed.

The object-oriented approach to software engineering originated in the domain of program design and implementation. The successful use of this approach has resulted in the adoption of object-oriented technologies in other domains, each domain deriving different benefits from the various aspects of the technologies. Examples of such domains are databases, artificial intelligence, distributed systems and systems management.

This exploration of object-oriented technologies first describes them in their original context of object-oriented program design and implementation. There then follows a discussion of the various possible benefits of object-oriented technologies in that context. X/Open's strategy towards object-oriented technology must take into account the impact of the technology on both databases and distributed systems. These two areas are examined next. Finally, there is a description of the various groups that are seeking to standardise aspects of object-oriented technologies.

1.1 OOT for Program Design and Implementation

This section examines fundamental object-oriented concepts in the context of software development employing object-oriented programming languages. Such object-oriented software development has two elements. Firstly, there is an approach to the analysis and design which allows the construction of good models of problem domains. Secondly, the design is implemented in a programming language which permits the direct expression of the features of these models. The benefits of using object-oriented technologies arise from the nature of these models, whose key features are their use of three concepts: *encapsulation*, *inheritance* and *polymorphism*. An understanding of these concepts will provide a suitable basis for an understanding of object-oriented technologies as they are currently used in commercial environments. A more general summary of object-oriented concepts can be found in the referenced Essence of Objects article.

1.1.1 Encapsulation

The problem domain is decomposed by identifying its key abstractions, hence discovering classes of objects with common state and behaviour. For each class, an external interface is defined; this is separated from the details of the implementation. Each object is then an instance of some class and its clients can access the object only through the well-defined interface. The programming language will allow the expression of this encapsulation of the object, hiding the implementation details from clients of the object.

1.1.2 Inheritance

A common feature of complex systems is that classes of objects tend to have some aspects of their state or behaviour in common, while remaining identifiably different:

- Two classes may have identical interfaces but differ in semantics or implementation. For example, a Stack and a Queue might have an interface which simply offers the ability to store and retrieve elements and return a count of their contents; the interfaces are identical but the implementation and semantics differ in important details.
- One class might offer all the facilities of a particular class but also some additional interface. For example, a Window class might have an interface to allow it to be moved and resized. A Scrolling Window class would also offer those facilities but add the ability to scroll the contents.

It is the ability to model such relationships that gives object-oriented technologies much of their power. Typically, these relationships are expressed using inheritance. Inheritance can be used to express commonality of interface or commonality of behaviour.

In the first example, the common features of Stacks and Queues may be abstracted by defining a general Container class. The Stack class and Queue class are then seen as particular examples of the general Container. The Stack class and Queue class inherit their interface from the Container class. This interface inheritance is a valuable tool for expressing design features and, as shown in Section 1.4 on page 10, is exploited extensively when object-oriented principles are applied to distributed systems.

Considering the second example above, in the design the Scrolling Window would be derived from the Window class; the intention being that it would offer the Window class' moving and resizing capability but would also offer additional functionality. The metaphor of inheritance is again used; the Scrolling Window class of objects inherit aspects of their behaviour from the Window class. Hence the Scrolling Window class is described as a child of the Window class and the Window class is described as a parent of the Scrolling Window class.

As an aside, the terms subclass and superclass are used by some authors to describe these relationships. The terms are considered to be confusing and will not be used here. The reasons for this decision are described more fully in the referenced Object-Oriented Software Construction document (page 233). In summary, a Scrolling Window can be seen as being both less than and more than a Window, being a special case and yet adding additional functionality. The meanings of sub and super in this context are then ambiguous.

One obvious intention of the designer in identifying inheritance relationships between classes of objects would be to avoid the duplication of the code implementing shared behaviours. When implementing the class, an object-oriented language would not only allow the relationship between the two classes to be expressed directly, but would also, by virtue of expressing that relationship, have requested that the common code be reused.

The following C++ example demonstrates this approach; some syntactic detail has been omitted for the sake of clarity. This first fragment is a declaration of the Window class interface offering move and resize functionality.

```
class Window {
public :
     move(int Rows, int Columns);
     resize(int Rows, int Columns);

// etc.

};
```

The Scrolling Window class then simply reuses that existing interface and adds its own additional functionality.

```
class ScrollingWindow : public Window {
public :
     scroll(int Lines);

// etc.

};
```

The implementor of the Scrolling Window class then simply needs to write the code for the scroll function in order to complete the class. This is an example of what has been termed *differential development*. The benefits of this approach will be discussed later.

The designer may also recognise a further aspect of an inheritance relationship between two classes of objects: a child class may offer some adaptation of its parent class' behaviour. In the case of the ScrollingWindow the *resize*( ) operation might well perform all the operations required for the resizing of a Window, but also perform additional special processing for the scroll bars. This design decision can also be represented in an object-oriented programming language. The following example shows how such a construct would be represented in C++.

```
class ScrollingWindow : public Window {
public :
     scroll(int Lines);

     resize(int Rows, int Columns)
     {
          // Call the parent's resize function
          Window::resize(Rows, Columns);

          // and special code for scrolling windows here
          // etc.
     }

// etc.

};
```

1.1.3 Polymorphism

The final major feature follows from the concept of inheritance. The concept is that an instance of a child class, by virtue of offering all of the features of the parent class, can be used wherever an instance of the parent could be used. In the example above, a Scrolling Window *is* a Window; that is, it offers all the functionality of a Window. Therefore, in places in the design, some general operation can be defined to operate on Windows, and in design terms this operation acts not only on any object of the Window class, but also on any child such as the Scrolling Window. Again the object-oriented programming language allows this concept to be expressed.

This feature is more powerful than it may at first appear. In the example above, the Window class would have an interface to allow the window to be drawn. The Scrolling Window class would also have such an interface; however, it would presumably need to differ in some ways from the Window class. The design would presumably intend that given any object of a Window class or child class, the correct version of the draw function would be invoked. An object-oriented programming language would allow the economic expression of this concept.

In the Window example, the function *draw*( ) is added to both classes, along with an indication in the Window class that child classes may implement their own versions of the function.

```
class Window {
public :
     move(x, y);
     resize(x, y);

     virtual draw();
          // "virtual" is an indication that children will
          // have their own versions of this function
};

class ScrollingWindow : public Window {
public :
     scroll(x);
     draw();
};
```

A function can then be written which operates upon objects of the Window class via a pointer to such objects. The Scrolling Window class is a child of the Window class and so the function will

operate transparently on either Windows or Scrolling Windows, and the invocation of the draw function will cause the corresponding version of the function to be called.

```
SomeFunction( Window *pWindow)
{
     pWindow->draw();

    // etc.
}
```

The function *SomeFunction*( ) may then be described as taking an argument that may be of a number of different types; that is, a pointer to a Window or a pointer to any child of a Window. This is an example of polymorphism, a literal translation of the term being *many-shapedness.* It should be noted that the function is type safe; pointers to objects of classes not derived from the Window class may not be passed to this function.

This functionality may appear to be little more than a syntactic convenience. It does, however, lead to a number of positive benefits which are among those discussed below.

1.1.4 Object-Oriented Programming Languages

The relative merits of the many programming languages supporting the expression of object-oriented design are debated with considerable fervour by some proponents of object-oriented software development. In the appendix to the referenced Object-Oriented Design document, Booch identifies more than 100 such languages and also provides a useful basis for their comparison. A further discussion of some more recently developed object-oriented languages may be found in the referenced Dr. Dobbs journal.

The following list identifies those languages which seem to be of particular relevance to this report:

Smalltalk — Smalltalk originated in the early 1970s at the Xerox Palo Alto Research Centre and its concepts have influenced the design of many other object-oriented languages. It is seen as ‘‘pure’’ object-oriented language; even integers and classes themselves are objects. Experts in the field of object-oriented programming commend Smalltalk as a good vehicle for learning object-oriented concepts.

C++ — This language was designed by Bjarne Stroustrup of AT&T Bell Labs. It is largely a superset of C and was originally implemented by means of a preprocessor for C known as *cfront*. The language directly supports object-oriented programming and also permits the coder to ‘‘drop-down’’ to procedural programming in C.

Eiffel — Eiffel was developed by Bertrand Meyer and is discussed in the referenced Object-Oriented Software Construction document. In addition to supporting the expression of object-oriented programming constructs, it also has features intended to improve software reliability. The key concept as stated by Meyer is ‘‘*programming by contract*: the relationship between a class and its clients is viewed as a formal agreement, expressing each party’s rights and obligations’’. Eiffel allows the direct expression of pre-conditions, post-conditions and class invariants. Eiffel seems to have attracted its greatest degree of support in Europe.

Objective-C — Like C++, Objective-C is an extension of C; it adds features and syntax of Smalltalk to C. The language is described in the referenced Object-Oriented Programming document. The language is of special interest in that it has been used by the NeXT corporation to construct their NeXtStep environment.

1.2 Claimed Benefits of OOT

Having identified the key differentiators of object-oriented technologies, it is now possible to examine the claimed benefits of their use. Information gathered in the interview process (see Section 2.1 on page 18) enabled an evaluation to be made of the effectiveness of object-oriented technologies. That information is presented in Chapter 2; the current chapter presents the benefits without attempting to evaluate the claims.

1.2.1 Management of Complexity

Software is increasing in complexity both in the visible functionality it offers and in the problem domains it addresses. Programs become both larger and more intricate. Object-oriented design techniques help the developers to manage larger and more complex problems. A number of reasons are suggested:

- The design vocabulary open to the object-oriented designer is appropriate for modelling complex problems. In particular, the specification of inheritance relationships amongst classes of object uses a rich metaphor not explicitly supported in technologies that are not object-oriented.
- By encouraging encapsulation, object-oriented technology allows the problem domain to be considered in manageable pieces.
- The ability to use a differential approach to specification also allows the problem space to be modelled by a process of gradual refinement.

1.2.2 Adapting to Change

Object-oriented technologies tend to ease the adaptation of software to changing requirements. All three of the key features described above contribute to this:

- Encapsulation reduces the coupling between system components, and reduces communication by side-effect, making the dependencies that do exist between such components explicit. The result is that ripple-effects of change can be greatly reduced.
- Inheritance provides a mechanism whereby considerable leverage is obtained from existing code when additional or modified functionality is required.
- Polymorphism allows new classes created via inheritance to be added without affecting existing designs and code. Consider the *SomeFunction*( ) example above. If some new class is derived from Window, then instances of that class are processed by *SomeFunction*( ) without any design or code change. Compare this with how procedural code such as this might need to be modified:

```
SomeFunction(int WType, void * pWindow)
{
     switch(WType)
     {
          case WINDOW:
               drawWindow((Window*)pWindow);
               break;

          case SCROLL_WINDOW:
              drawScrollingWindow((ScrollingWindow*)pWindow);
              break;

          /* ... etc. */
     }
}
```

Not only would such code need to be extended for each new Window type, but so would all other such functions. This situation should be contrasted with the object-oriented code presented earlier which requires no change when new derived classes are introduced.

There is also an additional reason why maintenance and enhancement are eased: fewer lines of code need to be written in object-oriented programming languages. This can be seen when comparing the versions of *SomeFunction*( ) above. In general, the conciseness of expression is not at the expense of the legibility of the code.

We touch here on the question of whether the use of object-oriented programming languages is actually necessary in order to obtain the benefits of an object-oriented approach to software development. Proponents of object-oriented techniques would claim that an object-oriented approach to analysis and design is in itself valuable, so some of the desired benefits may well accrue irrespective of the implementation techniques used. It is also acknowledged that it is possible to implement an object-oriented design in a procedural programming language. However, the practicality of such an approach is questioned by Booch in the referenced Object-Oriented Design document (page 36): ‘‘From a theoretical perspective, one can fake object-oriented programming in non-object-oriented programming languages ... but it is horribly ungainly to do so.’’ Of the three main features of an object-oriented design, encapsulation is the least difficult to represent in a procedural language. In fact, encapsulation is commonly seen as a standard ‘‘good practice’’ by experienced programmers in procedural languages. The difficulty arises when implementing inheritance relationships and polymorphism; these concepts do not have a natural expression in procedural languages. A somewhat overstated analogy would be that implementing inheritance in C is like implementing structured programming in early dialects of Basic; the required structure can be imposed, but language offers no enforcement of the desired idiom.

The author’s opinion is that generally the use of a language which is able to express object-oriented design features naturally is essential if the full benefits of an object-oriented approach are to be achieved. One practical compromise is offered by the Object Management Group architecture which is discussed in more detail later. This architecture permits an object-oriented interface to be defined in an Interface Definition Language (OMG IDL) and for the implementation of that interface to be in either object-oriented or procedural code. One intention of this is to permit legacy code to be accessed via an object-oriented interface. The result is that, irrespective of the actual implementation, from the perspective of the client of the interface an object-oriented language (OMG IDL) is being used.

1.2.3 Facilitating Development

It is claimed that the use of object-oriented development techniques generally facilitate the development process, with productivity and quality gains being achievable.

Productivity gains may come from two sources. Firstly, as discussed above, object-oriented programming languages are more expressive, and hence more can be achieved with less code. Secondly, there is the enabling of one of the most commonly perceived objectives of object-oriented software development: design and code reuse.

An important pre-requisite of code reuse is Meyer's open/closed principal. This is discussed in the referenced Object-Oriented Software Construction document (page 23). The concept is that code that is to be reusable needs to satisfy two apparently contradictory constraints. It must be *open*; that is, available for change and extension, because requirements will change as new users of the code attempt to use it. It also must be *closed*; that is, its interfaces must be stable otherwise clients will not be able to use it. Encapsulation (hiding details of the implementation), inheritance (allowing differential specification and partial reuse) and polymorphism (enabling existing code to act on newly introduced classes) all contribute to achieving these two conflicting goals.

A second type of code reuse is also supported by some object-oriented programming languages: the concept of generic code. The concept is that some algorithm, for example, a stack or B-tree, is defined in terms of a parameterised data type. Instances of that algorithm can then be obtained for some chosen data type. The chosen data types may include user-defined object classes. The effect is that complex algorithms can be implemented once and then readily reused in a type-safe manner.

1.3 Object-Oriented Databases

A very common application requirement is that the data being manipulated is stored on disk or some other persistent medium. Applications developed using object-oriented programming languages have a range of possible mechanisms by which this can be achieved. Methods used when developing in procedural languages are applicable to object-oriented software, so it is entirely feasible to store object data in file structures of arbitrary complexity. For example, a simple sequential file could be used or some indexing scheme might be employed. Another possibility is to store the data in a database; indeed there are a number of commercially available products providing interfaces between object-oriented languages and relational databases.

For many applications, a relational database will offer significant advantages over the direct use of the file system by providing facilities such as:

- concurrency control
- transaction management
- distribution of data
- backup and recovery
- query languages.

From the perspective of object-oriented software development, traditional relational databases and file systems share one deficiency: the application developer must accept responsibility for transforming the object-oriented data structures in his application into a form suitable for storage. These data structures will typically exhibit complex relationships between objects. Object classes will inherit from object classes, objects will contain objects, objects will refer to objects. It is entirely possible to transform the object data in such a way that it can be saved and restored; however, the transformation logic required may be complex. Further, the data, as saved to persistent storage, may not intrinsically describe the object relationships, the logic for the restoration of the data lying in the application.

Object-oriented databases (OODBs) are intended to allow a seamless integration of database objects and the objects as manipulated by the program. The database and the programming language can share the same type system, the database fully representing the inheritance, containment and referential relationships between the objects, while also providing the database services such as concurrency control and transaction management mentioned above. A more detailed discussion of issues of object persistence may be found in the referenced Making Objects Persistent article.

It should be noted that several relational database vendors are evolving their products to better accommodate object technology. Also there are proposals to add object extensions to the next revision of ANSI SQL. In consequence, the boundaries between object databases and relational databases are becoming less distinct.

An issue that will need to be considered is the degree of consistency of object models. It was explained above that the major facility offered by OODBs is that they permit a consistent model to be used for database objects and objects manipulated by the programming language. It appears to be probable that the same degree of consistency will not be seen between SQL object models and object-oriented programming languages. In particular, encapsulation and polymorphism may not be supported by SQL to the same extent that they are supported by OODBs.

The Object Data Management Group is a consortium of object database vendors. The group has produced object database standards for data definition, data manipulation and querying. These are defined in the referenced Object Database Standard document.

1.4 OOT and Distributed Systems

There is an increasing use of object-oriented techniques for the development of distributed software. Distributed systems management is one notable domain where the object-oriented techniques have gained widespread acceptance. The following sections consider the relevance of the concepts of encapsulation, polymorphism and inheritance to distributed systems. A more detailed discussion of the general applicability of object-oriented technology to distributed systems may be found in the referenced Distributed Computing Systems article.

1.4.1 Encapsulation

By considering a system as a set of objects each with a well-defined interface, encapsulation facilitates distribution in a number of ways. Firstly, by promoting modularity it helps to define suitable units of distribution. Secondly, the well-defined interfaces map well to message-based communications. Thirdly, the separation of interface and implementation lends itself to both location transparency and to the support of platform-specific variants of implementation in heterogeneous networks.

The separation of implementation and interface has a further implication for system developers who need to interwork with legacy systems. An object-oriented interface may be specified, but the details of implementation are concealed behind the interface. So the implementation may be in a procedural language, and may indeed be provided by some existing legacy system.

1.4.2 Inheritance

The specification of distributed objects can make considerable use of interface inheritance. Such inheritance permits commonality of interfaces to be recognised. Inheritance allows suppliers of distributed object components to specify portions of their interfaces in terms of common building blocks.

For example, an interface for a concurrent object could be defined as:

```
class Lock {
     BOOL ReadLock();
     BOOL WriteLock();
     void FreeLock();
     // ...
};
```

and an interface for a persistent object might be defined to include:

```
class Persistent {
     int SaveState();
     int RestoreState();
     // ...
};
```

A supplier of an Employee object is able to define an interface which offers concurrency and persistence in addition to some Employee-specific operations. The interface is composed by inheritance from the Lock and Persistent classes:

```
class Employee : public Lock, Persistent {
     String GetName();
     void SetSalary(Number);
     // ...
};
```

The following points should be noted:

- If a standard interface for concurrency and persistence is used by suppliers of objects such as Employee, then the task of the user of these classes is greatly simplified.
- The implementor of each derived class is not constrained to adopt any particular implementation of the inherited interfaces. Varieties of implementations are possible depending upon the relevant engineering constraints.
- The implementation of the object with this composite interface need not employ inheritance from some Lock and Persistence classes. The implementation might in fact use a non-object-oriented programming language.

1.4.3 Polymorphism

Distributed object interfaces support polymorphism in much the manner described in Section 1.1.3 on page 4. That is, an interface which is specified to accept as a parameter an object of some class, will transparently accept other objects of some derived class. The benefits of extensibility described in Section 1.1.3 on page 4 then clearly apply to distributed objects; an interface will be able to operate upon objects whose types were not known at the time at which that interface was constructed.

The Information Technology industry generally appears to have accepted the applicability of object-oriented technologies to distributed systems, and one major indicator of this is the progress of the Object Management Group. The Object Management Group (OMG) is an international trade association with in excess of 350 member corporations. There is considerable overlap of membership between the OMG and X/Open, and the two organisations cooperate closely; for example, the referenced **CORBA** specification is published jointly by the two organisations. The OMG specifications are clearly of great relevance to X/Open and they are discussed in the following sections.

1.4.4 OMG Goals

A detailed description of the goals of the OMG may be found in the referenced OMA guide, but in summary they are as follows. The OMG is dedicated to maximising the portability, reusability and interoperability of software across distributed heterogeneous platforms. The OMG intends to exploit the benefits of object-oriented technology and, in particular, the extensibility facilitated by the use of this technology.

There is a vision of the OMG specifications defining an architecture which permits the building of applications based on distributed interoperating objects, the applications presenting standard interfaces and hence being able to interoperate and being open to extension, and the fundamental building blocks being objects which provide services as specified by the OMG.

This architecture would have two major infrastructure elements: it must define general mechanisms by which objects communicate, and it must define the specific interfaces of the objects that are the primitive building blocks.

1.4.5 Object Communication

The OMG addresses the question of object communication in the specification of the Object Request Broker (ORB) in the referenced **CORBA** specification. This specification has been adopted by X/Open and has the standing of an X/Open Preliminary Specification. Implementations of this specification are available from a number of vendors, and other vendors are in the process of adapting their proprietary implementations of similar functionality to conform to the specification. By the end of 1993 it was possible to obtain CORBA-conformant ORBs from a number of sources including Digital Equipment Corporation, Hewlett-Packard Company, Hyperdesk, IBM Corporation, Iona Technologies, Isis Distributed Systems and SunSoft Inc.

The most important feature of the CORBA specification is its Interface Definition Language (IDL). The interface offered by an object is specified in this language. In principle, it is possible to map OMG IDL to many different programming languages. At the time of writing, only the C-language mapping has been published by OMG and a C++ language mapping is imminent; there is an intention to provide mappings to Smalltalk, COBOL and Ada. In fact, products offering proprietary implementations of C++ and Smalltalk mappings do already exist. Client code is then written in some particular programming language, using that language's expression of the OMG IDL interface. The implementation of that interface may also be written in one of the supported languages, the specific intention being that implementations are not required to be themselves object-oriented, thus permitting integration with heritage code.

The Object Request Broker then provides the mechanism whereby a client request is processed by the desired object. A wide variety of possible ORB implementations is possible. In a simple case, the client request amounts to little more than a direct invocation of a routine in a library. In a more general case, the ORB can provide complete location transparency in a heterogeneous network. Sophisticated ORB implementations are envisaged which support performance enhancing features such as replication of objects and load-balancing migration of objects between nodes in a network.

1.4.6 Object Services

The second element that the infrastructure must provide is a sufficient set of services accessed via the ORB so that useful work can be done. OMG has described an architecture for these services in the referenced OSA document. A distinction is made between low-level *Object Services*, the fundamental building blocks for application objects, and the higher-level *Common Facilities* which may provide direct end-user functionality.

Examples of Object Services are:

- event notification, allowing notification of events to interested objects
- persistence, allowing an object to be placed in persistent storage such as a file system, database or repository
- concurrency control, controlling multiple access to the same object.

An example of the higher-level Common Facility might be a generic help facility.

Both these categories of object are seen as being general purpose and reusable.

OMG is initially seeking to standardise Object Services. Four sets of such services have been identified. Recently, OMG announced the adoption of a specification for the first set of services. This specification is known as the Common Object Services Specification (COSS). It is based on a joint proposal submitted as a result of collaboration between 21 companies including Sun Microsystems Inc., Groupe Bull, NCR Corporation, Digital Equipment Corporation, Hewlett-Packard Company, ICL, IBM Corporation and major object-oriented database vendors. Work is

underway on the second set of object services, and there are indications that the OMG is also ready to commence work on the specification of Common Facilities. A likely first task is the production of a document-linking interface.

1.4.7 Future OMG Specifications

In addition to the ongoing work of producing specifications of Object Services and language mappings for C++ and other languages, the OMG is also seeking to produce a second version of the CORBA specification. This specification will seek to define a standard for interoperability between Object Request Brokers.

One important feature of the work of the OMG is that its standards are based on technologies that already exist. OMG technology represents the convergence of proprietary products, a number of which are already commercially available. Therefore, although a complete specification of OMG technology is still somewhat distant, there is opportunity for early use of these technologies.

1.5 Current Standards Landscape

There are a large number of organisations participating in the development of *de jure* and *de facto* standards and specifications of object-oriented technologies. Some groups are accredited subcommittees of standards organisations, others are consortia of industry organisations. Some impression of the extent of standards activities which in some way are related to object-oriented technology may be seen in the referenced Workshop Report, which lists some 40 separate initiatives by various organisations.

The following sections describe a number of these organisations which are producing standards and specifications that are not specific to a particular problem domain and whose activities are likely to be relevant to X/Open.

ANSI Committee X3J16/ISO WG21: C++ Language

This joint ANSI/ISO committee is producing a standard for the C++ programming language. The standard is based on AT&T reference documents for the language, but will include some additional features. The standard will include a small number of commonly used library components including I/O streams and Strings.

A working draft of the standard is scheduled to be available by the end of 1994.

ANSI Committee X3J17: Smalltalk Language

This committee is producing a standard for the Smalltalk programming language.

ANSI Committee X3H2: Database Language SQL

This committee is seeking to develop SQL beyond the current SQL-1992 standard, and is intending to include object extensions. The **SQL3** standard is scheduled for delivery in 1995.

ANSI Committee X3J4: COBOL

This committee will produce a revised COBOL standard whose review process is scheduled for completion by 1997. This standard will incorporate object extensions to COBOL. The proposal for the details of these extensions is expected to be approved by X3J4 and the corresponding ISO group SC22/WG4 by the middle of 1994.

Object Management Group (OMG)

This group is an industry consortium whose membership has a significant degree of overlap with that of X/Open. Its central purpose is to create a standard which realises interoperability between independently developed applications across heterogeneous networks of computers. Their standards employ an architecture based on distributed interoperating objects.

It should also be noted that the work of ANSI committee X3T3 (Open Distributed Processing) and the reference model produced by the joint ISO and IEC ODP sub-committee (ISO/IEC JTC1/SC21) group are relevant to the domain of distributed interoperating objects. The latter group also obtains leverage from European-based ANSA and ECMA initiatives.

The work of the OMG is discussed in more detail in Section 1.4.4 on page 11.

Object Database Management Group (ODMG)

This group is a consortium whose members include many of the leading vendors of object-oriented databases. They are seeking to standardise Data Manipulation Languages and Data Definition Languages for access to object-oriented databases.

The initial specification produced by this group is published in the referenced Object Database Standard document.

X3H7: Object Information Management

ANSI committee X3H7 was created in response to a recommendation by the Object-Oriented Database Task Group (OODBTG) of the Database Systems Study Group (DBSSG), one of the advisory groups to the ASC / X3 Standards Planning and Requirements Committee (SPARC).

The committee has a wide brief in the field of Object Technology, but has chosen to focus on the various object models in use by the industry, firstly with a view to understanding and describing these models, but also with a long-term objective of seeking their convergence.

Other Groups

In addition to these organisations, there are many groups which are applying object technology to particular problem domains. Examples of such activities are as follows:

- The X / Open Systems Management Programme is of special interest in that it represents the work within X / Open that has been most greatly impacted by object-oriented technology.
- SEMATECH is a U.S. non-governmental consortium of companies concerned with semiconductor manufacturing. The consortium is seeking to apply object-oriented technologies to Computer Integrated Manufacturing.
- ANSI committee X3H4 seeks to provide standards and facilities to specify, integrate and manage an enterprise's information resources and assets.
- ANSI committee X3H6 seeks to produce models for CASE Tool integration.

However, despite the many significant contributions made by the above groups, the development of current object-oriented technology is still at a stage where rapid evolution is taking place. The pace of change is such that *de facto* standards have yet to emerge, and the process of producing *de jure* standards is likely to be overtaken by events. The tension between the market's desire for the clarity promoted by standards and the need to avoid the stifling of innovation by premature standardisation is evident.

Chapter 2

Experience, Needs and Opportunities

A pre-requisite for action by X/Open concerning object-oriented technologies is that there should be evidence that object-oriented technologies are, or will be, important to the open systems market. X/Open's strategy will be further affected by any particular market trends. Opinions concerning these matters were obtained from a series of interviews; the following sections describe the interview process, content and findings.

The information is presented as follows:

- a discussion of the participants in the interview process and the interview content
- an examination of the current use of object-oriented technologies by the companies whose representatives were interviewed, the activities of their customers, and a number of other indicators as to the current state of the object-oriented technologies marketplace
- an evaluation of the degree to which the promised benefits of object-oriented technologies are actually obtained, having drawn together the various experiences of the interviewees
- in order to set the scene for a consideration of possible X/Open strategies, an examination of future possibilities for developments of the object-oriented technologies; this section considers both needs arising from current obstructions to the effective use of object-oriented technologies, and opportunities which are presented by new technologies
- some conclusions drawn from the material above.

2.1 The Interviews

This chapter is based on information gathered from interviews with representatives of a number of X/Open member companies.

2.1.1 Interview Participants

Invitations to participate were made to X/Open Technical Managers who, in some cases, chose to nominate representatives with some special interest in object-oriented technologies. The expectation was that some three or four member organisations would offer representatives. It is a significant indicator of the status of object-oriented technology that the level of interest was such that the following eight companies were willing to provide representatives:

- Digital Equipment Corporation
- IBM Corporation
- Hewlett-Packard Company
- Informix Software Inc.
- NCR Corporation
- SunSoft Inc.
- Transarc Corporation
- Unisys Corporation.

Additional interviews were conducted with representatives of NeXT and Taligent — two organisations which are not members of X/Open but who have a strong commitment to object-oriented technology.

The set of people interviewed were, almost exclusively, actively involved in the use and development of object-oriented software. Further, in most cases there was a commitment to OMG technology, with many representatives being closely involved in the production of CORBA-conformant products. Given this homogeneity of background and evident belief in object-oriented technologies, it is unsurprising that there was a considerable uniformity of opinion.

2.1.2 Interview Content

Prior to the interviews, a discussion paper was issued to those Technical Managers who had expressed an interest in the study and to their nominated representatives. This paper also outlined the proposed set of questions for the interviews.

The interviews were intended to address the following issues:

- How and why are object-oriented technologies being deployed?
- What are the current and likely future market responses to object-oriented technologies?
- What specifications are needed to define an open, multi-vendor environment for object-oriented software development and execution?
- What role should be adopted by X/Open in this context?
- How, and to what extent, should X/Open specifications evolve to meet the needs of the open systems community in the light of the developments in object-oriented technology?

This chapter concentrates on the first three of these issues. The remaining two have provided input for the evolving X/Open OOT Strategy and specifically for the 1994 Technical Programme.

2.2 Current Deployment of OOT

2.2.1 Use by Represented Companies

The people interviewed fell into three categories, the largest group representing object-oriented software development arms of major hardware vendors providing open systems solutions. A second group, Taligent and NeXT, represented organisations committed to the production of complete object-oriented software environments, while a third group, Informix Software Inc. and Transarc Corporation, represented independent software vendors whose primary products are not aimed at the object-oriented software market.

The following points emerged:

- As might be expected, those software developments producing object-oriented products do themselves employ object-oriented development techniques.
- More significantly, a number of representatives see object-oriented techniques gaining increasing acceptance as being applicable to software development in general. For example, major software developments use object-oriented design techniques even though the eventual coding, for compatibility reasons, will employ a non-object-oriented language.
- Object-oriented developments are not limited to smallscale one and two-man projects; even excluding the large developments of Taligent and NeXT, team sizes of 15 and larger were mentioned.
- Object-oriented developments are not limited to experimental status. Teams have been established for several years, and products have progressed beyond initial releases.
- The deployment of object-oriented technology is limited by the existence of large bases of existing non-object-oriented code.
- The language used for development of production software is almost exclusively C++. There is some use of Smalltalk, although it is more frequently in the context of the construction of prototypes. Eiffel is not in common use amongst those interviewed; it is, however, possible that European-based contributions might have shown a greater use of this language.

A number of benefits resulting from the use of object-oriented technology are commonly seen:

- Speed of development was cited, especially in contexts where requirements were fluid.
- Both encapsulation and inheritance are seen as promoting maintainability.
- It was felt that complex requirements were more easily satisfied when using object-oriented design techniques. The object-based decomposition of the problem is found to be helpful in managing complexity.
- There is also a feeling that a use of object-oriented technology tends to result in a greater emphasis on the design process rather than coding, with consequent beneficial side-effects of improved quality.

There is also agreement on a number of factors which need to be accommodated when object-oriented technologies are to be adopted:

- There is a genuine paradigm shift in moving from procedural software development to object-oriented development. In this transition, the learning of a programming language is seen as the minor component; the focus needs to be on the learning of object-oriented design techniques.
- A common sentiment is that, despite its dominant position, C++ is not embraced enthusiastically. Two major criticisms of C++ are offered. Firstly, it does not enforce an

object-oriented approach; it is easy to slip back into a procedural approach. For this reason, it is seen as a poor vehicle for the learning of object-oriented concepts. Secondly, it is a complex language that is both hard to implement and hard to use; there are subtle traps for the unwary user. Some indication of the nature of these traps is given in the referenced Guessing Games article.

- The immaturity of object-oriented development environments was frequently mentioned. This is manifested both in instability of the C++ language and its inconsistencies of implementation, and also in the lack of supporting tools such as class browsers and standard class libraries. This situation both impacts productivity and hampers the production of portable code.
- In principle, it should be possible to greatly ease software development by assembling an application from objects selected from libraries from a variety of sources. In practice, there are currently technical and logistical difficulties in doing this.
- The large existing procedural code base limits the growth in use of object-oriented technologies in many of the organisations.

2.2.2 Use by Customers

The general consensus of those interviewed is that customers are beginning to adopt object-oriented technologies, with many large organisations having prototype or pilot schemes in place, and a number having completed the development of significant applications. There are also particular application areas where object-oriented development is well established; Computer Aided Design (CAD) and Financial Systems being two contrasting examples. The driving force for using object-oriented technologies is different in the two cases.

An object-oriented model is seen as being particularly appropriate for CAD, and it is found that implementation of such models is eased by the use of object-oriented databases which allow the complex data structures to be readily stored in a manner consistent with the underlying model.

The motivation for the use of object-oriented technology in building financial systems is that system requirements tend to be complex and fluid. The benefits of ease of maintenance and support for differential programming are then highly applicable.

Suppliers have released products facilitating object-oriented development, and there is a general commitment to continuing this process. The general belief is that use of object-oriented technologies will continue to increase. When timescales were discussed, a picture was painted of object-oriented software development becoming widely accepted for production software development within five years. It is clear that, within the IT industry as a whole, different market segments will adopt object-oriented technologies at different rates. It is likely that many commercial organisations whose current expertise is in COBOL-based development will wait for ANSI standard implementations of COBOL-with-objects before adopting these technologies. In contrast, it appears that much mainstream Personal Computer-based software is already being developed using object-oriented languages such as C++.

2.2.3 Information from Other Sources

Consultants' Conspectus

The details of a recent small survey conducted in the U.K. are discussed in the referenced Consultants' Conspectus document. This survey disclosed a pattern consistent with the picture described above.

- Many of the respondents had yet to investigate object-oriented techniques. Further, and somewhat surprisingly, a number of respondents were unfamiliar with the concepts of object-oriented technology.
- The majority of those using object-oriented technology are doing so in small projects involving less than 10 man years of effort, typically in some form of pilot study.
- The dominant language used is C++, with Smalltalk being the only significant alternative.
- The transition to the use of object-oriented techniques was found to be difficult.

Survey by Taligent

A description of a survey carried out by Taligent can be found in the referenced Taligent White Paper. The participants in this survey were 14 major companies who are using object-oriented development techniques for business-critical applications. The fact that such a survey was possible does in itself indicate that object-oriented technologies have gained significant acceptance.

Again, a picture emerges that is consistent with those described above; there are a number of details of interest:

- The dominant language in use is C++, with Smalltalk being used to a lesser degree. One interesting comment was that companies who attempted to use more than one programming language found that interoperability challenges inhibited design and code reuse. It is interesting to speculate as to whether the integration architecture offered by a CORBA implementation would be accepted as a solution to this problem.
- The management of transition to the object-oriented paradigm was again seen as a critical success factor. It was accepted that an investment in training and a readiness to learn from people with experience of object-oriented technologies were essential.
- The iterative nature of object-oriented development was emphasised. This point is also brought out in the referenced Object-Oriented Design document (Chapter 7). Three points emerge. Firstly, iterative development with frequent end-user involvement tends to produce systems which meet end-user requirements. This benefit is also mentioned in the referenced Consultants' Conspectus document. Secondly, the facilitation of change offered by object-oriented technologies underlies this approach. Thirdly, object-oriented design and analysis is difficult. An expectation that robust, reusable designs can be got right first time is over-optimistic.
- The objective of design and code reuse is hard to achieve, not only because of the technical difficulty of producing reusable code, but also because of problems of logistics and group dynamics. It may be hard to find existing code applicable to a particular problem; it may be hard to overcome the ''Not Invented Here'' syndrome.

Other Indicators

There are a number of indications that object-oriented techniques are gaining widespread acceptance. In particular, the C++ language appears to be supplanting C as the systems programmer's language of choice.

- All major MS-DOS and MS-Windows-based C compiler vendors have replaced their products with C++ compilers. These products do also offer C compilation facilities, but their focus is support for C++, and in some cases sophisticated additional toolsets for C++ programming are supplied. The MS-Windows versions of these compilers are supplied with substantial C++ class libraries and application frameworks.
- Technical publications now make increasing use of C++ as the language in which to supply example code. One example in particular may be significant: the C User's Journal now devotes at least 50% of its editorial material to C++.
- In the author's local book shop, the 15 different titles available directed at the C developer were massively outnumbered by the 30 different titles for the C++ developer.

2.3 Evaluation of OOT Effectiveness

The overall view of the interviewees is that the object-oriented technologies do at present offer real, achievable benefits. As will be discussed later, it is also felt that there are further benefits yet to be fully realised.

The general picture given by the interviewees is that initial experiences with object-oriented technologies are found to be sufficiently positive that the users of these technologies will adopt them for further and increasing use. The relative ease with which various benefits were achieved is discussed below.

2.3.1 Maintenance and Enhancement

The most frequently mentioned realised benefit of using object-oriented techniques was that of the flexibility of the software constructed. Such software is seen as being more readily extensible and maintainable. It was also frequently stated that object-oriented techniques are especially suitable in situations where requirements are fluid or ill-defined, because of the resulting flexibility of the software.

Practical experience appears to justify the claims described in Section 1.2.2 on page 6, in particular:

- Encapsulation has the effect of limiting the effect of change. The tendency of small changes to ripple throughout a product is reduced. One particular manifestation of this is that the implementation of an object can be modified to satisfy some changed requirement without modifying its interface and hence without affecting the implementation of any of its clients.
- The use of inheritance is found to greatly ease the extension of existing code. Incremental change is facilitated.

2.3.2 Speed of Development

Object-oriented software development is popularly expected to enhance programmer productivity. This expectation has been realised in the experience of some of the people interviewed; however, the general consensus is that considerable investment in staff training is required before significant gains are achieved. Typically, any productivity gains are masked by such start-up costs in early projects.

2.3.3 Code Reuse

Another popularly supposed benefit of the use of object-oriented techniques is the support for code reuse. There are three aspects of code reuse to consider:

- Reusing code as it is developed is generally not found to be easy. It requires specific management input to ensure that it happens, and this is especially the case when project teams are large. However, processes such as design and code reviews which may encourage the identification and adoption of common code will have other benefits. Hence, the cost of achieving the goal of code reuse may be offset by additional quality gains.
- As an organisation makes greater use of object-oriented techniques, it will tend to accumulate libraries of code that are specific to its problem domain. It is when such libraries are mature and available within an organisation that much of the code reuse payoff occurs. This is a further example of how the full benefits of the use of object-oriented techniques tend not to be seen in early projects.
- It should be remembered that code reuse is in fact already widespread throughout the mainstream software industry, in that a large number of generally useful system libraries are

available. C and C++ programmers have access to very large amounts of reusable code on both open and proprietary platforms. As an example, consider the graphical and networking code reused when an X interface is invoked.

The additional value that object-oriented techniques offer is that the mechanisms for such reuse are improved. The major requirement for code reuse is that a stable interface should exist. The historic contribution made by X/Open to the open systems community is its publication of specifications for such stable interfaces. The encapsulation offered by object-oriented specifications offers an improved way of specifying a stable interface.

It must be emphasised that designing object-oriented code which can successfully be reused is a non-trivial task. The twin goals of providing a class that is useful, stable and encapsulated and yet also open to unforeseen extension are not readily met. An iterative style of development is proposed by many experts in the field of object-oriented technology.

It would then seem essential that any class libraries proposed for standardisation should have undergone significant live use and be tried and tested.

2.3.4 Management of Complexity

The expectation that object-oriented design techniques would be suitable for handling large and/or complex problems does seem to have been realised in the experience of a number of interviewees.

2.3.5 Design Quality

An interesting side-effect of the use of object-oriented technology was that there seemed to be greater focus on design phases of the lifecycle, and this had a beneficial effect on the quality of software produced.

2.4 Enabling the Development of OOT

The previous sections have described a view of the current state of object-oriented software development derived largely from the direct experience of the representatives interviewed and the reported experiences of their customers.

One common theme that emerged was that there is a belief that the full benefits of object-oriented technology have yet to be seen; object-oriented technologies are still young technologies. Many of those interviewed have visions of the future directions in which object-oriented technology would develop, and the shape of the software development landscape that would result.

There were two aspects to these visions. Firstly, there is a belief that various aspects of the infrastructure required to support object-oriented software development are currently lacking. It is generally accepted that there is sufficient momentum behind the use of object-oriented techniques for these problems to eventually be addressed. A number of these issues are examined below. Secondly, there was a belief that the OMG architecture, in addition to providing a good mechanism for the development of distributed applications, offers a real prospect for the synergistic interworking of software components from multiple vendors. Some general comments on the use of OMG specifications are presented below.

The common elements of the various visions were that, in various guises, software reuse should be possible to a much greater extent than at present. Elements of these visions are already achievable in an *ad hoc* manner on some platforms. However, coherent support across open systems platforms is currently more problematic, and the situation is even less clear when OS/2 and MS-Windows are also considered.

2.4.1 Infrastructure for Object-Oriented Development

The following sections describe a number of issues concerning the current degree of support for object-oriented software development. Each of these issues reflects some obstruction to the effective use of object-oriented technologies. There is a strong bias towards C++ in these issues, reflecting the fact that the vast majority of object-oriented developers are using that language. By far the most commonly cited obstruction to the more effective use of object-oriented technologies was the immaturity of the current toolsets; it will be seen that this theme underlies most of the issues listed below.

Language Standardisation

The C++ language is complex and at present no *de jure* standard for the language exists. ANSI committee X3J16 and ISO committee WG21 are currently working to produce such a standard. The two committees hold all meetings together and produce a single set of documents.

For compiler vendors, the C++ language has represented something of a moving target. A comparison of the language as described in the referenced Stroustrup 1st Edition document and the referenced Stroustrup 2nd Edition document shows that major features were added to the language in a relatively short time. Two such features, a systematic way of handling exceptions and templates which are an implementation of the generic specification facility described in Section 1.2.3 on page 8, are not yet available in the C++ compilers available from some major vendors. Further, there are subtle differences of interpretation of language features between different compiler vendors.

X3J16/WG21 effectively base their work on the referenced ARM document, which annotates and comments on the language description in the referenced Stroustrup 2nd Edition document. However, the committee will address a number of further language extensions in addition to resolving current ambiguities. A summary of the issues considered can be found in the

referenced C++ Standardisation article, and an indication of which extensions are likely to be adopted by X3J16/WG21 is given in the referenced Recent Language Extensions article, although at least one extension not mentioned in that article has since been adopted.

This somewhat confused landscape will begin to stabilise once the X3J16/WG21 definition of the language becomes clear. A working draft of the standard is scheduled to be produced by the end of 1994. It is likely, therefore, that significant inconsistencies between C++ compilers will remain at least until that draft is produced.

Despite the general lack of consistency of language implementations on UNIX-based platforms, there is some degree of conformance. This is largely because many implementations are based on AT&T's *cfront* technology. In comparison, the major vendors of C++ compilers on MS-DOS and MS-Windows have significantly different implementations, some offering templates, and some offering exception handling.

This rapid evolution and lack of consistency of interpretation is a cause of concern to potential users; however, it should be noted that this has not prevented the widespread adoption of C++.

Standard Class Libraries

A further obstruction to the use of object-oriented technology is the lack of standardisation of class libraries. This problem is exhibited at two levels. At the low level, there is a need for fundamental classes for manipulating quantities such as files, strings, dates, matrices and also for object containers such as stacks, queues and associative arrays. At a high level, there is the potential for constructing class libraries that offer powerful functionality, the most common example being libraries of classes which allow rapid construction of graphical user interfaces.

Compiler vendors typically supply class libraries offering some version of the low-level functionality, and these versions typically have significant differences in both function and interface. The effect is, to quote the chairman of X3J16/WG21 in the referenced C++ Standardisation article, ''... to cause almost every development project using C++ to develop their own additional libraries''. This problem was mentioned in the interviews as being commonly experienced by customers. The problem is addressed to a limited extent by X3J16/WG21, which will standardise a small set of low-level functions including file access and strings. Other low-level classes were considered to be too complex and diverse for standardisation at present.

When higher-level constructs are considered, the developer is faced with a large range of divergent products from many vendors. Typically, the various higher-level class libraries offered will each provide some version of the low-level functionality. Effectively, there is a choice of interface to which to write, with no guarantee that a chosen interface will be available on all target platforms. Some products attempt to offer a unified approach to writing for various graphical interfaces such as X, MS-Windows and Presentation Manager. Others are focussed on one particular such system.

The general opinion of those interviewed was that the state of the technology was such that it would be premature to attempt to standardise on any one such class library, it being preferable to wait for convergence brought about by market forces.

Combining Class Libraries

Further problems face the C++ developer at present when attempting to mix-and-match libraries from multiple origins.

Firstly, there is no standard for the interworking of the executable form of C++ classes. It is not generally possible to intermix class libraries developed with different compilers. This tends to lead to the distribution of libraries in source code form.

A second problem then arises from the lack of common base classes. Libraries from different sources may well contain different implementations of the same or similar low-level functionality. This, in the more benign case, leads to bloated executable sizes; in the less benign case, there are namespace collisions so that the libraries cannot interwork. This latter problem will ultimately be addressed by an extension to the language adopted by X3J16/WG21.

Compiler Technology and Supporting Tools

Historically, the logistics of software development, especially when large teams of developers are involved, have always been problematic. A typical problem has been the compilation costs incurred when data structure definitions and macro definitions change. When C++ is used, these problems are potentially exacerbated; small changes can require disproportionate amounts of recompilation. While, with care, the software developer can avoid such problems, it is possible in concept for the development environment to give a much greater degree of support by offering facilities such as incremental compilation.

Another area commonly mentioned was the need for tools which enable the browsing of class library facilities. Sophisticated integrated development environments offering such facilities are becoming available, and it is believed they will offer real benefits.

It is interesting to note that relatively few of the organisations contacted were making use of object-oriented CASE tools. The general theme here was that these tools were immature or not well suited to system software development.

Class Library Size and Complexity

A developer working in a mature project environment will have access to several class libraries: some general purpose, perhaps encapsulating access to a relational database or offering facilities to build graphical user interfaces, others addressing particular problem domains. Some of these libraries will have been developed by third-party suppliers, others developed in-house. In some cases class libraries will have been developed which refine the behaviour of third-party classes to provide some required behaviour.

In principle, the rich set of reusable software available should provide considerable help to the developer. Development should largely consist of assembling preconstructed components into the desired form. In practice, this is not readily achieved unless the developer is familiar with the structure of the available classes and their inter-relationship. A view was expressed that code reuse was sometimes not achieved because it seemed to be quicker to rewrite code than locate and understand existing code.

2.4.2 OMG and Object Interworking

In considering the future, the representatives were all supportive of the OMG specifications and saw significant benefits arising from the use of its facilities for object interworking. Although the OMG specifications have a general application to the construction of distributed software, they also may be useful in a non-distributed environment. In particular, the specifications provide a common interface for objects developed in arbitrary languages and used by clients also developed in arbitrary languages, hence they address the problem of interworking class libraries described in **Combining Class Libraries** on page 27.

Various comments and questions were raised about the OMG specifications. These are outlined below.

Continued Evolution

The specifications are, as yet, immature and incomplete. The OMG has work in progress to complete the major elements that were seen to be missing from the specifications: a C++ language mapping, object services and ORB interoperation. The major elements of these specifications are likely to be seen before the end of 1994. It is very likely that, as experience is gained with the use of implementations of these specifications, extensions and refinements will be made.

In the meantime, it should be remembered that CORBA-conformant products are already available in the industry and that OMG activities represent convergence of existing technology.

Problem Domains

No consistent view emerged of the general applicability of the CORBA interface, with the major concern being performance characteristics. Some doubt was expressed about using a CORBA interface for small, frequently accessed objects: there were fears that overheads of marshalling data and security might be unacceptable. Other reservations concerned the OMG decisions of which actions were to be services accessed via an ORB, and which were to be directly offered by the ORB.

In principle, the implementors of ORBs and object services have considerable freedom of implementation, and so the fears expressed above may not be realised.

Use of OMG IDL

There were a range of views about the use of OMG IDL for interface specification. In addition to concerns about performance, which may be a question of quality of implementation rather than being intrinsic to the use of OMG IDL, there was concern that in offering support for many languages, the support for each is compromised to some extent. In the case of C++, two powerful language features are not supported. Firstly, there is no support for the C++ template feature which allows a generic interface to be specified using parameterised types. Secondly, there is no support for using the C++ ability to differentiate the elements of a class interface exported for public use and that exported for use by child classes. Given the current dominance of C++, this was seen as unfortunate.

However, the language independence offered by OMG IDL was considered to be significant for two reasons:

- There was consensus that the large amount of heritage code extant in the industry would result in the continued use of procedural languages, in particular C, for many years. The C-language bindings to OMG IDL are therefore significant, allowing both existing functionality to be encapsulated and offered via an ORB, and also the opportunity for C-language clients to make use of object services.

- There was a general lack of enthusiasm for C++ and a perception that it will be necessary to facilitate the use of other languages. The potential for developing other language bindings is therefore important.

2.5 Conclusions

Some general conclusions about the state of object-oriented technologies may now be drawn:

- The use of object-oriented technologies on open systems platforms is already established and will continue to grow. It is expected to become the dominant force in the development of new software within 5 years. The implication is then that applicable open specifications would be helpful at present, and that within 2 years there will be a pressing need for such specifications.
- Object-oriented approaches will not supplant procedural techniques within the next 5 years. The existence of large amounts of heritage code will require that the X/Open CAE specifications continue to be supported on open platforms. It is also probable that the high-level object-oriented libraries and frameworks used by object-oriented developers will at least in the short and medium terms be implemented using CAE facilities. The existing CAE specifications will retain their value for the foreseeable future.
- There is already considerable confusion in the marketplace due to the plethora of consortia and standards and specifications bodies. The open systems community must seek to reduce this confusion. The repeated message from the interviews was that X/Open must not attempt to duplicate the efforts of other groups as this will tend to increase confusion. Further, the same key players are members of many of these bodies. If input about the same issues is required in many contexts, excessive duplication of workload can result in a diffusion of effort and, in the worst case, a lack of consistent input.
- Several of the representatives mentioned an issue that arises from the concept of software reuse that is rarely seen in popular discussions of object-oriented technologies: this is the issue of quality.

  To overstate the case, an understandable attitude can be: ‘‘Whose software am I prepared to trust? Mine!’’ The problem is that if the industry realises the goal of software construction by assembling and refining third-party objects, then the responsibility for the quality of the resulting system becomes diffused.

  If developers are to be able to create objects conforming to some interfaces that are (using polymorphism) substitutable for existing objects, then they must be able to readily verify the quality of those objects.

Chapter 3

X/Open Future Directions on OOT

It seems certain that object-oriented technologies will continue to gain acceptance as primary tools for the development of software on both proprietary and open platforms, and that the technologies will be seen as facilitating the development of complex, flexible software and providing an infrastructure for the development of distributed applications. X/Open seeks to ensure that its specifications evolve, in breadth and scope, to support users of object-oriented technologies on open systems platforms.

The views expressed and the information derived during the creation of this study resulted in a set of specific recommendations for future X/Open activities. These will form the basis of X/Open's OOT strategy, initially through the creation of a Technical Programme for OOT within X/Open's overall 1994 Technical Plan. This plan will address the following specific areas:

Programming Languages
Application Environment
Object Databases
Object Interworking

An important goal of the OOT Technical Programme will be to maximise the mutual benefit obtainable through X/Open's collaboration with OMG in the area of OOT, and also to minimise unnecessary duplication of effort with other relevant organisations.

Glossary

class
A category into which objects are placed on the basis of both their purpose and their internal structure. A class lists the data relating to an instantiation of an object, and the methods that act on the data.

class libraries
Collections of classes regarded as having a general applicability.

CORBA
Common Object Request Broker Architecture.

encapsulation
Process of ensuring that data may only be accessed by a method. Classes define methods and data together.

IDL
Interface Definition Language.

inheritance
The construction of a definition by incremental modification of other definitions.

method
A subroutine or procedure that belongs to a class.

object
An object is an identifiable, encapsulated entity that provides one or more services that can be requested by a client.

object management
The creation, examination, modification and deletion of potentially complex information objects.

object services
Typically, low-level system services defining interactions between objects.

OOT
Object-Oriented Technology.

ORB
Object Request Broker. Provides the means by which clients make and receive requests and responses.

polymorphism
The ability of methods in different classes to use the same name: multi-named.

Index