



**Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom**

TELEPHONE:
INTERNATIONAL:
FAX:
E-MAIL:

**Cambridge (01223) 515010
+44 1223 515010
+44 1223 359779
apm@ansa.co.uk**

Training

ANSAwise - Objects in Distributed Systems

Chris Mayers

Abstract

Organizations will be aware of object technology, and have a picture of the benefits it can bring, but may be unaware of the state of the market, and the pitfalls.

This module of the ANSAwise training programme discusses “objects in general”, compares how the ideas of encapsulation, polymorphism, and inheritance are applied in various object technologies, and explains why distributed objects require a specific approach to be successful.

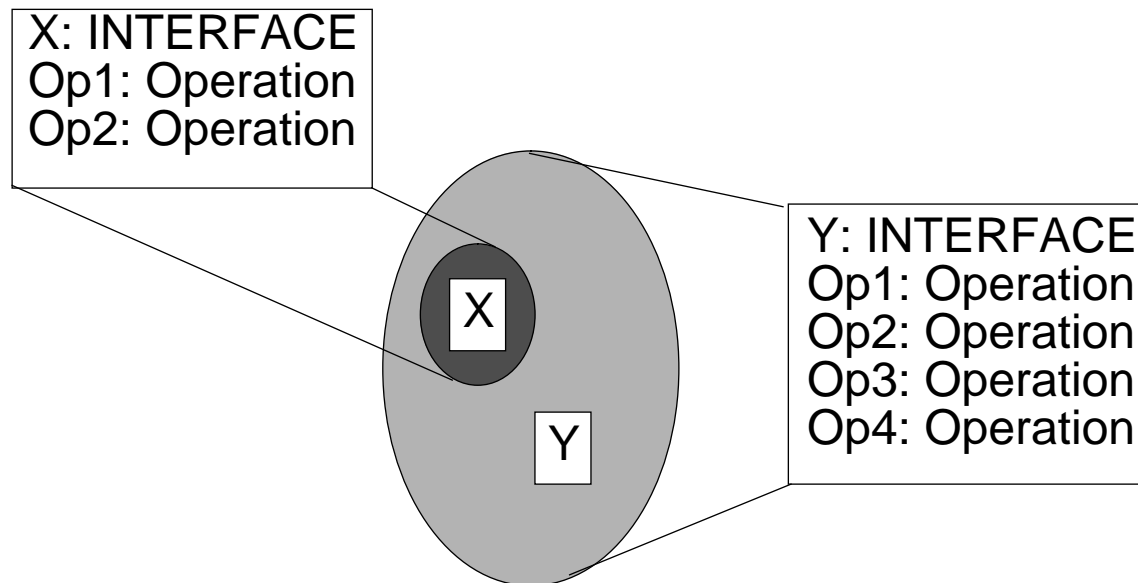
APM.1350.01

Approved
Briefing Note

28th November 1994

Distribution:
Supersedes:
Superseded by:

Objects in Distributed Systems





In this session

- *Explain the basic ideas of object technology*
- *Show how these ideas are applied in programming languages, databases, and development methods*
- *Examine some commercial systems*



Why object technology?

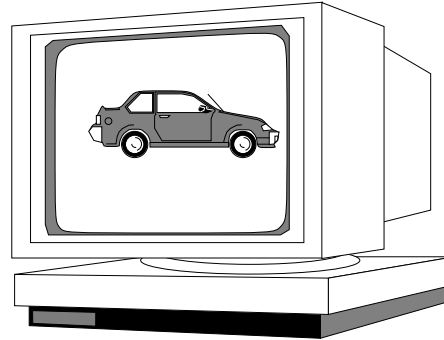
- *Object technology enables systems to be built out of modular components*
- *These systems offer*
 - **Greater productivity through reuse**
 - **Better quality through proven components**
 - **Lower costs through reduced maintenance overheads**



Influence of object technology

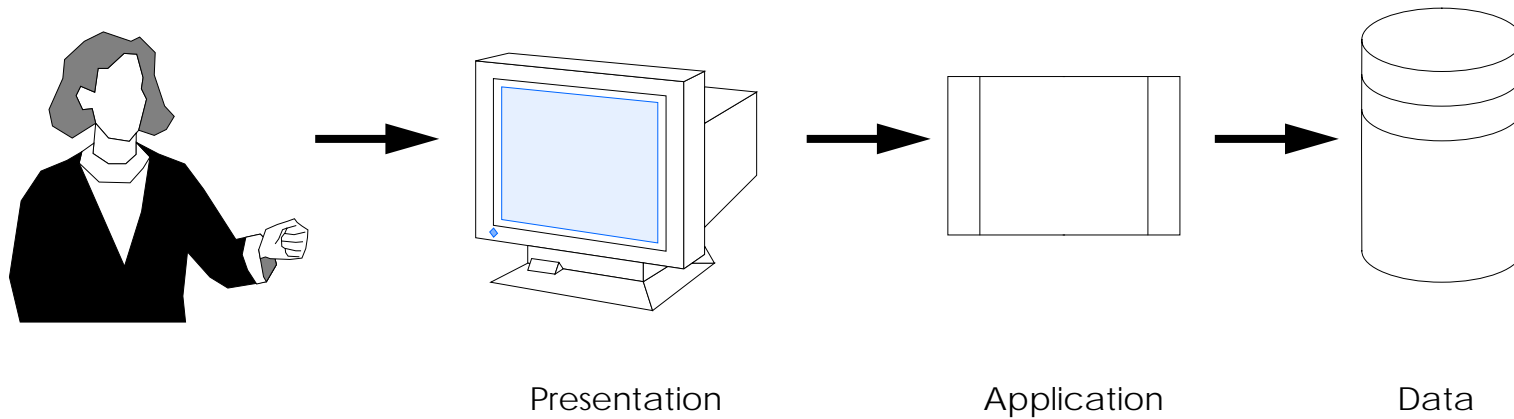
- ***Now affecting all areas***
 - operating systems
 - programming languages
 - databases
 - user interfaces
- ***Products and standards are emerging in all of these***
 - but at different levels of maturity...
 - ...with patchy support from development methods and tools
 - ...and with important technical differences
- ***Not all of these products and standards are 'open'***
 - this poses a challenge to those building systems with object technology

Seamless programming?



- *In a car rental system, the car may be represented as*
 - a user interface object, on the display
 - a distributed systems object, when accessed remotely
 - a database object, in the OODBMS
- *These objects are programmed using different tools, maybe even in different programming languages by different people*

Objects Everywhere





Business Objects and Technology Objects

- *Business objects are derived from a top-down analysis of your organization's goals*
 - derived from a strategic enterprise model
 - embodied in an operational model
 - ... with associated business rules
- *Business objects might be customers, products, orders...*
- *Technology objects are acquired bottom-up*
- *In the future, both business and technology objects may be bought and sold*
 - supplied by third parties in horizontal and vertical markets

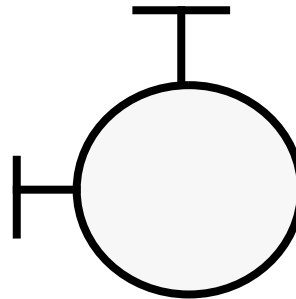


Objects - beyond modular programming

- *Object programming is 'modular programming' that scales*
 - to very large and complex systems
- *Object technology has three common concepts*
 - encapsulation
 - polymorphism
 - inheritance
- *In distributed systems these concepts need to be examined again*

Encapsulation for objects

- ***Objects are encapsulated***
 - every object provides a service via interfaces
 - the interface is public; the implementation is private and hidden
 - encapsulation forms a boundary; the only access to an object is via its interfaces



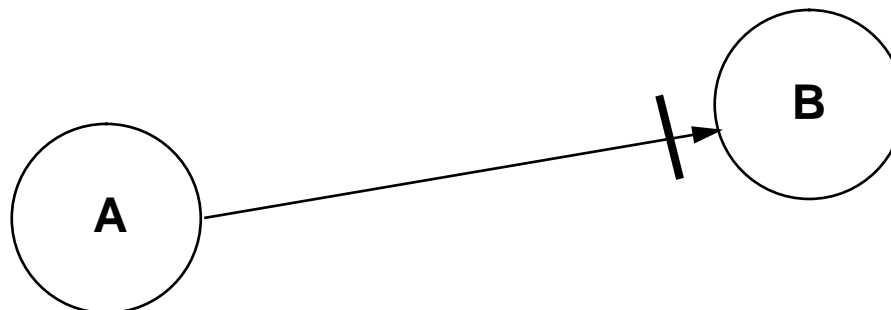


Encapsulation in programming languages

- *Access only via explicit interfaces*
 - C++ class
 - Smalltalk class
 - Ada package
- *Hide the implementation*
 - the data structures
 - the algorithms
 - the state
- *Keep the data with the code*

Encapsulation in distributed system

- *In a distributed system, encapsulation must be strict*
 - no shared state between objects
 - no 'back doors'
- *Objects A and B could be on the same machine today...*
- *... in different countries tomorrow*



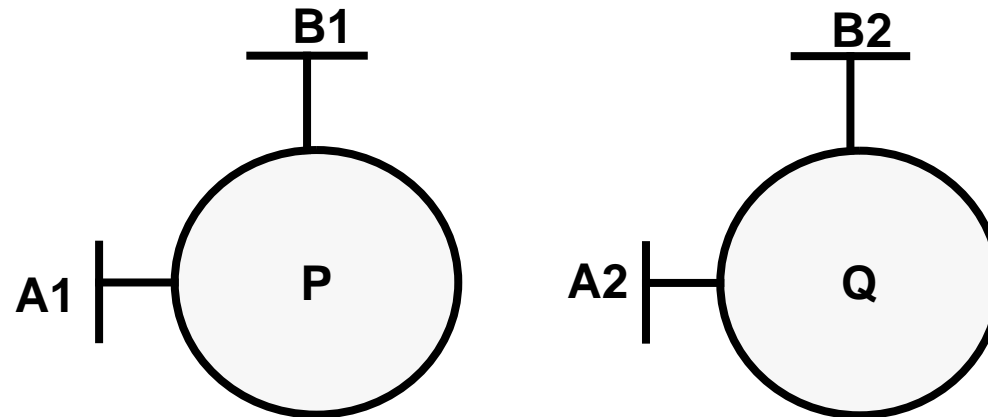


Polymorphism

- *The same operation can apply to different types of object*
 - you can print a text file, a structured document, and a bitmap
- *Each object can be expected to “know how to print itself”*
 - it has a standard interface for printing
 - the object determines *how* to “print itself”

Inheritance for objects

- *Consider two similar objects*



- *They can have similar interfaces A and B*
- *They can have similar implementations P and Q*
- *...these are quite separate properties*

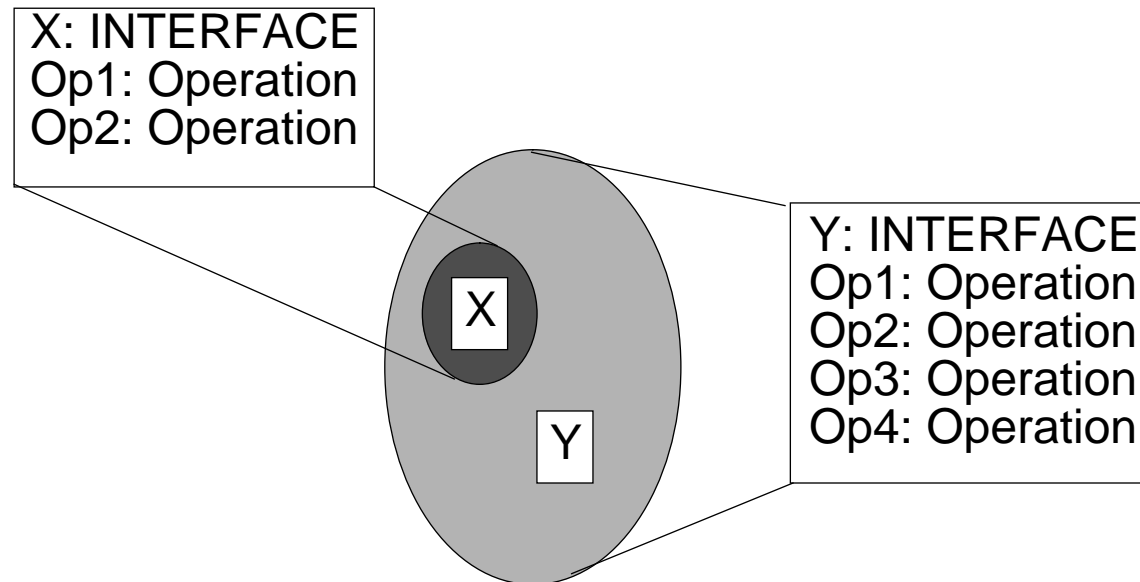


Views of inheritance

- *In non-distributed object systems we may be interested in similar implementations*
 - sharing or reusing some of the code...
 - ... for example the code for the A interface
- *In a distributed system this is not relevant*
 - sharing any code between distributed objects is impossible
- *The important thing is compatibility of interfaces*
 - type conformance (*substitutability*)
- *Some programming languages try to use 'class' for both ideas*

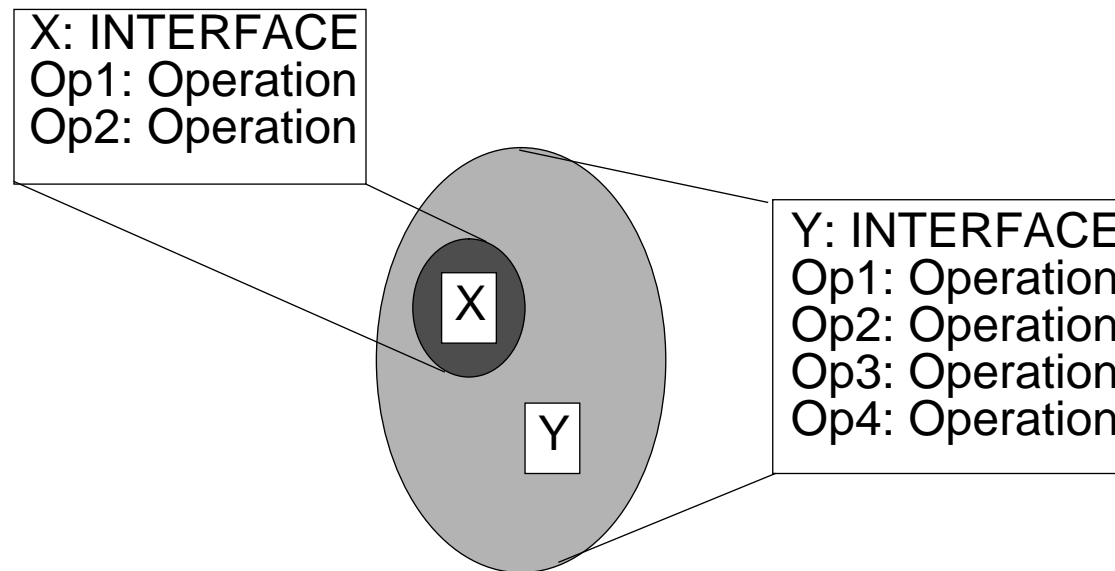
Type Conformance

- *Conforming interface types do not have to be identical...*
- *...Interface type Y conforms to interface type X*



Type Conformance and Subtypes

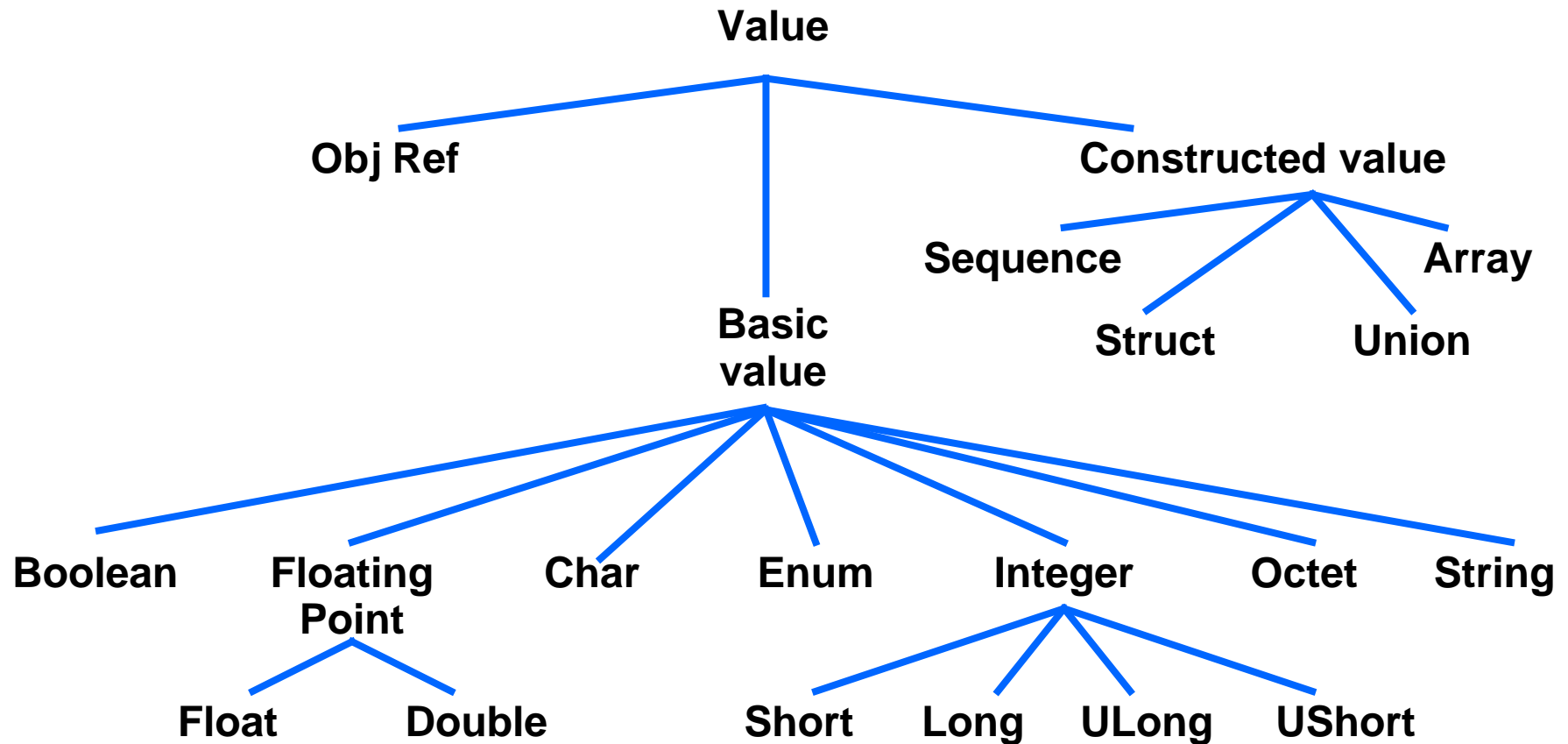
- *Be careful when comparing types*



- *Y is a 'subtype' of X...*
 - ... but Y can 'do more' than X, not less!



Be Careful of Type Hierarchies



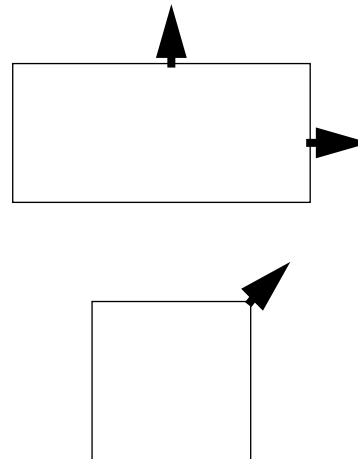


What type hierarchies can mean

- *When someone shows you a picture like that, it could have several meanings*
- *Does it show*
 - *type conformance relationships?*
 - *implementation relationships?*
 - *something else?*

'Is-a' relationships

- *A square 'is a' rectangle...*

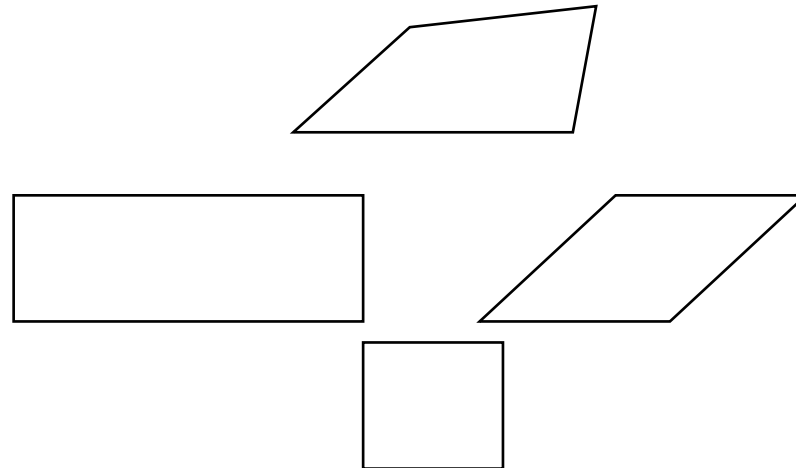


- *But their 'resize' operations do not conform*
 - rectangles have separate operations for resizing horizontally and vertically
 - squares have only one resize operation



Is-a relationships and type conformance

- *Type conformance does not match is-a relationships*



- *There is no 'natural' type hierarchy*
 - *all that matters is conformance*



Object lifetime

- *Objects outlive the processes that create them*
 - conventional pointers are inadequate...
 - ... a more general kind of 'handle' is needed
- *In a distributed system, it must be possible to pass these 'handles' around between machines*
 - their representations must be mutually recognized
- *Some systems have a global 'unique id' (UUID) for each object*
 - this is not actually necessary



Some Distributed Object Technologies

- *DEC ObjectBroker*
- *Microsoft COM (part of OLE)*
- *Component Integration Labs (CIL) OpenDoc*
- *Hewlett-Packard Distributed Object Management Facility (DOMF)*
- *IBM Distributed System Object Model (DSOM)*
- *Sun Distributed Objects Everywhere/Distributed Object Environment (DOE)*



Products and Technologies

- *Some are proprietary interfaces with standards-based lower layers*
- *Some are direct implementations of standards*
- *Some are products, with bundled services which may be proprietary or not proprietary*



Microsoft's Object Linking and Embedding (OLE) - user features

- *Compound documents*
 - for example, spreadsheet tables inside word processor documents...
- *Drag and drop*
 - ...select part of a spreadsheet table and drop into the workspace of another application
- *Visual editing (in-place activation)*
 - ...edit the embedded spreadsheet table without leaving the word processor

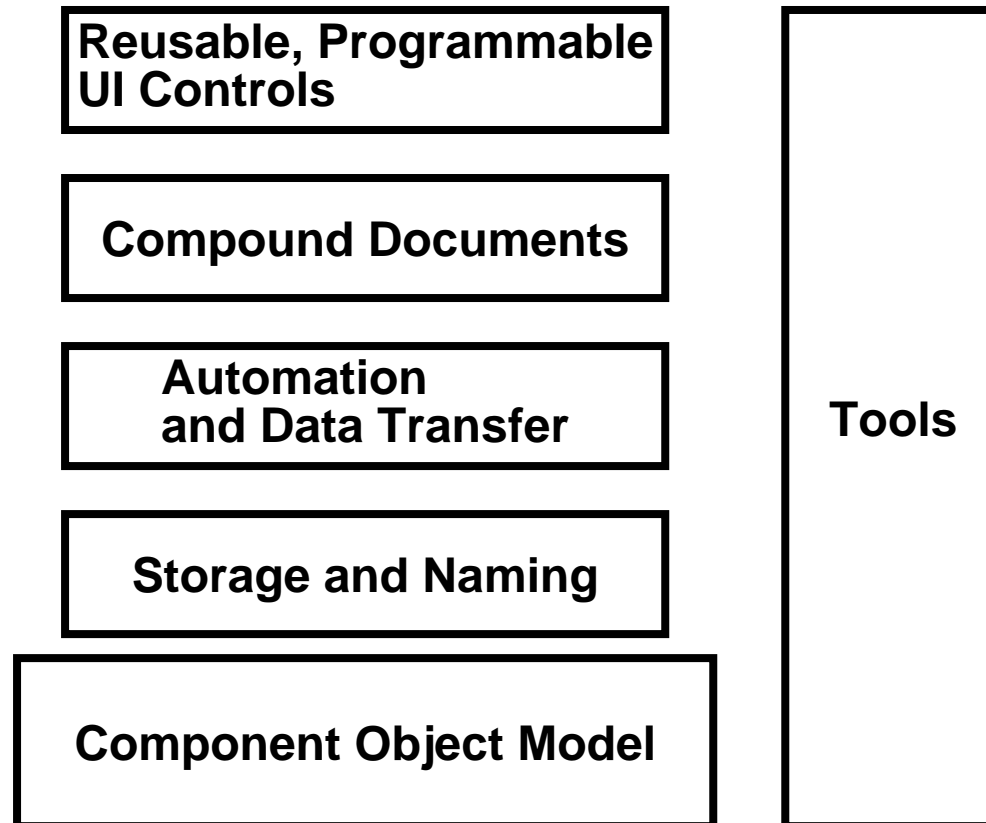


Microsoft OLE - application features

- ***Structured storage***
 - standard interfaces for applications to store and retrieve compound documents
- ***Automation***
 - applications and documents can be controlled by Visual Basic for Applications (VBA)
- ***Component object model (COM)***
- ***Uniform data transfer***



OLE Services





Microsoft COM and distributed systems

- *OLE 2 does not have support for distribution*
 - it uses LRPC (lightweight RPC) for communications; not networkable
- *OLE 2 has 'clients' and 'servers'*
 - but these are not networked
- *The Component Object Model (COM) underpins OLE 2*
 - COM will support heterogeneous networking
- *Microsoft and DEC are cooperating to allow their object systems to interwork*
 - using a specification called the Common Object Model...
 - ... helpfully, this is also called COM!
- *Distributed OLE will bring these technologies together (in 1995?)*



Component Integration Labs (CIL) OpenDoc

- *Originally developed by Apple*
- *Not tied to a specific user interface technology*
- *Based on CORBA*
- *Will interoperate with OLE 2*



Summary

- *The key concepts are encapsulation and type conformance*
- *Distributed systems allow you choose freely the front-end and tools technology*
 - Programming language
 - User interface/toolkit
 - Database
- *You can use object systems without an object programming language*
- *Distributed systems development is possible without objects*
 - but is harder work
- *For more on this topic*
 - see *Distributing Objects (TR.018)*