III

Objects as a Systems Organisation Principle

J. Brenner

Rapporteur: Macieji Koutny

Twenty-first University of Newcastle upon Tyne International Seminar
on Teaching of Computing Science at University Level

Objects as a System Organization Principle

by John Brenner,
ICL, Bracknell UK.
September 1988

1. Introduction

The subject of this lecture is Objects as a system organization principle, applied to distributed processing. This raises issues different from those in the more familiar fields of object-oriented programming languages and object-oriented database.

I will talk about experience in the ANSA project and in Open Distributed Processing (ODP) standardization in ISO and ECMA.

ODP standardization should have major technical and commercial impact in the 1990s, and hence an impact on the computer science curriculum.

We have found object-oriented techniques to be an essential ingredient of distributed processing architecture, and a source of great technical strength. But we have also encountered difficulties. These are, in part, cultural problems of different perceptions that people have of object-oriented techniques, not just technical issues. This may be of particular concern to educators.

What I am about to describe is not "yet another object system" laying claim to intellectual turf in this field. Rather, I intend it to be an account of what object-oriented concepts we have found to be useful in this ongoing work.

Consistent use of terminology is difficult to achieve. My general starting point is the terminology defined by Peter Wegner in [1]. I will point out differences where they arise.

2. Who we are

I would like to begin with a few words about the organizations involved.

ANSA is the think tank in which these ideas have been sorted out. It is a UK Alvey project for industrial exploitation of research results in the field of distributed processing. This is a collaborative project, in its fourth year. The Chief Architect is Dr. Andrew Herbert, formerly of the Cambridge University Computer Laboratory. The project team is located at Cambridge, and consists of about 12 people, mostly seconded from the Companies collaborating in the project (British Telecom, DEC, GEC, HP, ICL, ITL, Olivetti, Plessey, Racal). This work is expected to continue via an ESPRIT 2 project, with enlarged membership (the Integrated Systems Architecture project, ISA). A description of the ANSA project is in a draft Reference Manual [2], and there is a summary in [3].

ODP standardization in **ISO** (the International Organization for Standardization) started in early 1987, and has led to the formation of a working group (ISO/IEC JTC1 SC21/WG7) which is developing a Reference Model of Open Distributed Processing (ODP-RM). The ODP-RM will define an architectural framework for distributed processing standardization, within which relevant new and existing standards will be positioned. This new work reflects growing recognition in ISO that Open Systems Interconnection (OSI) [4], which has been the focus of attention in recent years, is only part of the distributed processing story. References [5] [6] and [7] provide general information about ODP standardization and its motivation.

ODP standardization in **ECMA** (the European Computer Manufacturers Association, of which most of the world's leading computer manufacturers are members) started several years earlier. ECMA has recently produced an RPC standard [8], is contributing actively to the ISO Reference Model work, and is pushing ahead with standardization of an ODP object support environment [9], positioned within the emerging Reference Model. ANSA is the main source of technical input to this ECMA work. Another major contributor is the IBM European Networking Centre at Heidelberg.

I am the convenor of this ECMA group (TC32-TG2), and also participate in the ANSA and ISO activities. I am employed by ICL, where I have a responsibility for distributed systems architecture. ICL produces a wide range of systems for selected markets, primarily European. A major ingredient of our business is integration of heterogeneous distributed systems; hence our interest in Open Distributed Processing standardization.

ANSA has gained informal technical leadership in ODP standardization work by participating actively and to good effect in ISO, ECMA and the UK standards body, BSI. We hope that this will result in ODP standards soundly based on current research results. To encourage this orientation ANSA contributed at the start of the ISO work a comprehensive survey and reading list of current distributed processing research, reproduced in [10].

3. Open Distributed Processing

The field in which we are applying object-oriented principles is Open Distributed Processing (ODP). I will briefly explain what ODP is about before going on to consider the object-oriented ramifications.

- **Open** means conforming to standards, such that products can be procured successfully from multiple independent competing suppliers. Such standards should be:

  - technically sound,
  - commercially practicable,
  - under public control,
  - applicable world-wide.

  These are difficult requirements to satisfy, not the least because much of the technical work has to be conducted as a prolonged struggle for world-wide consensus in international standards committee meetings. This is very different from working in your own laboratory, and is a difficult environment in which to achieve technical excellence. Those of us involved in these processes do the best we can. The experience is often frustrating, but on the whole enjoyable and productive.

- **Distributed** means consisting of logically separate components. They may also be physically separate, to degrees varying from close proximity to widespread geographical dispersion. An important special case is logically separate components that are co-located, not physically separate.

- **Processing** means computation and its mechanization.

Therefore, **distributed processing** is about computation achieved via multiple separate modular computing agents directed towards some common purpose; and **open distributed processing** is about standardization to facilitate industrialization of distributed processing.

The notions of "computation via modular computing agents" and "object oriented computing" have much in common, and this was our starting point for use of object-oriented techniques in ODP.

4. The Problem of ODP

We now take a closer look at the problem space to which we seek to apply object-oriented techniques. This part of my talk is based on an ISO document [7] which discusses the requirements for distributed processing and the motivation for developing ODP standards.

The distribution of information systems is a necessary consequence of constraints arising from the real world which affect *all* kinds of information systems. The field of application of distributed processing techniques is, therefore, virtually unlimited. Some diverse examples are:

- Data Processing Systems;
- Database Systems;
- Office Systems;
- Process Control Systems;
- Knowledge Based Systems;
- Integrated Data/Text/Voice/Image systems;
- C^3I Systems;
- Home Entertainment Systems.

The point to be made here is that this field includes requirements (such as fault-tolerant processing, real-time processing and interactions based on isochronous voice and image) that are not usually on the agenda when object-oriented techniques are evaluated.

Heterogeneity is another major ingredient of the problem space to be addressed by ODP:

- Computing equipment heterogeneity;
- Interconnection network heterogeneity;
- Operating systems heterogeneity (and almost none of the operating systems are object-oriented);
- Computational heterogeneity (different languages etc., usually not object-oriented);
- Authority heterogeneity (cooperation between separate organizations);
- Application heterogeneity (desire to integrated different kinds of applications together).

The point to be made here is that whereas most other work on object-oriented techniques assumes homogeneity (the usual subject is an object-oriented programming language, an object-oriented operating system, etc.), we have to use and evaluate object-oriented techniques in the context of extreme heterogeneity.

The essence of the problem space to be addressed by ODP is *distribution transparency* which is about managing the consequnces of separation. This has several different ingredients, which are needed to varying degrees in differing circumstances:

- **Access transparency.** Concealing the use of communications when accessing remote resources.
- **Location transparency.** Enabling the use of a resource, independent of the placement of that resource in the distributed system.
- **Migration transparency.** Enabling the migration or reconfiguration of resources in a distributed system.
- **Replication transparency.** Enabling the use of multiple instances of a resource for such purposes as enhancing dependability and performance.
- **Concurrency transparency.** Avoiding inconsistencies due to parallel execution, by using concurrency control techniques.
- **Fault transparency.** Concealing faults by using error processing techniques.
- **Performance transparency.** Minimizing the performance penalties associated with using remote resources.
- **Scaling transparency.** Concealing variations in system behaviour due to scaling up to large or busy or turbulent systems, and scaling down to small or placid systems.

The point to be made here is that the core of our technical work is about combining distribution transparency techniques with object-oriented techniques. Again, this is a major distinction from other usage of object-oriented techniques.

One further point is that ODP must consider distributed information systems from many viewpoints other than that of the computer scientist. Ideally, a common core of object-oriented techniques would provide unifying abstractions applicable to a distributed processing system as viewed by its users, managers, designers, programmers, operators, maintenance staff, etc.

To summarize, major considerations when selecting, adapting and using object-oriented techniques for ODP are:

- Diversity of applications;
- Heterogeneity;
- Distribution transparencies.

These are the same kind of considerations that would apply to any approach to the integration of large scale software systems (i.e. programming-in-the-large).

5. Framework of abstractions

The ODP-RM will be primarily concerned with *modelling* and *specifying* the structure of distributed information systems. We need some agreed framework of abstractions within which to deploy our object models etc.

The current proposals for this framework are documented by ISO in [11]. The purpose is to identify sets of abstraction with which distributed information systems can be described for ODP purposes. The emerging consensus is that five sets of abstractions are appropriate. These are currently labeled A-E (we are still arguing about their names).

A. Abstractions for *enterprise* information modelling, defining what the information system is required to do for the enterprise concerned.

B. Abstractions for the *information* structure and the information flows of the system design.

C. Abstractions for the operational and *computational* aspects of the system design.

D. Abstractions for the *engineering* design that supports the distributed processing.

E. Abstractions of the *artifacts* with which the system design is realized (heterogeneous operating systems, computers, communications, etc.).

Each set of abstractions provides a vocabulary to produce a *closed world projection* of the actual information system being modeled. **Closed world** in the sense that all relationships are between things in that projection, and there are no references to things outside it. **Projection** in the mathematical sense that descriptions in different projections are all complete descriptions of the same system, viewed in different lights. The formal basis for this kind of modelling is taken from J.F. Sowa [12]. Independent corroboration of five generally applicable sets of abstractions is provided by J.A. Zackman in [13].

The acid test is: does this framework of abstractions work for us ? Many of us think that it will, and there is some practical evidence to support this. But the matter is not yet finally settled.

I do not have time now for more detailed description and justification of this framework. Two key points relevant to this lecture can be drawn from it:

- **Common Modelling Technique.** To achieve consistency across these different projections, we require that one general-purpose modelling technique is used throughout. The choice of that technique is what this lecture is about.
- **Focus of ODP Standardization.** We are now agreed that projections B (information structure), C (computational), and D (engineering to support distributed processing) are the core subject area of ODP standardization, most especially D.

It has also been made clear the the ODP-RM has no pretensions to be "the architecture of the universe of information systems". It is only intended to be "the architecture of distributedness" (which is still an ambitions undertaking). Figure 1 summarizes our current understanding of this framework (*note that projections do not imply layers*).

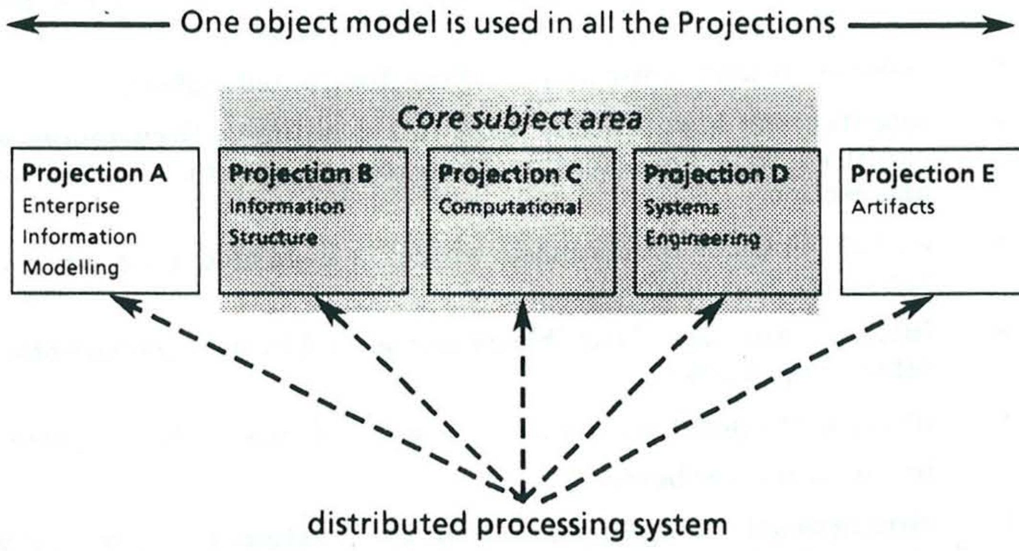


Figure 1: The Framework of Abstractions.

This overview of ODP has set the scene for considering Objects as a system organization principle applied to distributed processing.

6. Modelling Technique Requirements

An ISO working paper [14] explores the requirements for modelling in the ODP Reference Model. It was drafted in June of this year by a group of experts with different technical backgrounds and from several countries. The principal author is Professor Peter Linnington of the University of Kent, and it draws heavily on material provided by Andrew Herbert (ANSA) and Elspeth Cusack [15] of British Telecom. It is summarised in this part of the lecture.

The kind of modelling technique needed is explained step-by-step. As anticipated, the answer is an object model. But because the term "object" is heavily overloaded by different assumptions and different meanings, we have avoided using it in the initial exploration.

The first step of this exploration is clarification of what we mean by "modelling technique".

- A **model** is a representation of a system simplified for the purpose of description, simulation or calculation.
- A **modelling technique** is any technique used to construct a model. As such it is equivalent to a distinct language.
- Our concern is **specification**, hence a specification language.

- A **specification language** is a language tailored to the expression of *requirements and properties* (as distinct from an implementation language concerned with mechanisms). It may include graphical and textual formalisms.

The next step is to identify concepts to be expressed in a specification language for distributed processing. An incomplete list is:

- **component:** an arbitrary part of the distributed system.
- **occurrence:** a significant point in space time. Occurrences will be captured in a specification in terms of events at some level of abstraction.
- **event:** a set of occurrences which is regarded as atomic in the specification concerned.
- **interaction:** a set of possible events shared by all members of a given set of components.
- **interaction point:** the set of locations associated with an interaction.
- **inheritance:** see below.
- **configuration:** specification of the system in terms of which components interact, and at which interaction points.

We can consolidate this into an object model in which:

- In general an **object** is whatever is the subject of description.
- An ODP-object is an **abstraction** by which arbitrary *components* of a distributed system are modeled in a *specification language.*
- ODP-objects are characterized by the **interactions** in which they can participate, defined in terms of *shared events.*
- The defining abstraction for such interactions is an **interface specification.**
- Specifications are **organized** into families by *inheritance* structure.

This exploration confirms that an *event-oriented object model* is needed. This finding is controversial, in that some ISO participants with a database background expected to use exclusively an object model based on information modeling techniques (entity, attribute, relation, etc.).

7. Inheritance

Inheritance is widely recognized as an essential characteristic of object-oriented languages; see [1] and [15].

An inheritance structure is a hierarchy that defines the way in which objects are classified into families, such that the properties of an object are deemed to apply also to objects subordinate to it in the inheritance hierarchy. The inheritance hierarchy provides the basis for re-use and reproducibility of design, and the substitution of alternative implementations of objects while maintaining conformance with required characteristics.

In an object-oriented specification language for defining ODP architectural structure, the concept of inheritance necessarily applies to *inheritance of specification*. Any inheritance of implementation mechanisms (e.g. re-use of code), however desirable, is an implementation matter outside its scope.

In ODP we have adopted an algebraic framework for inheritance, defined in set-theoretic terms by Elspeth Cusack of British Telecom. This defines inheritance as a hierarchy consisting of a set with a partial order, an equivalence relation, a binary operation and conditions satisfied by that set. The details are in [14] and [15].

Related choices are that ODP usage of object-oriented inheritance will be:

- **Derived from an unique top element.** This ensures that all members of an object hierarchy can be related (the above mathematical model allows more than one top element).
- **Strict inheritance.** The exclusion of "non-strict" inheritance removes a source of complications which threaten the crucial re-use, reproducibility and substitution characteristics. Also we do not have a mathematical model for non-strict inheritance.
- **Multiple inheritance.** This increases the expressive power of the specification technique, and helps with object composition and decomposition. It is covered by the mathematical model.

These decisions relate to selection of object-oriented concepts for ODP architecture. They do not preclude use of other object-oriented techniques (e.g. other styles of inheritance) in the languages etc. used to implement distributed processing systems.

That is about as far as we have got in the ISO deliberations on object-modelling in Open Distributed Processing.

8. Where Next?

In this part of the lecture I will give a preview of what ANSA and ECMA are likely to feed into ISO over the next year.

Intension and extension. ODP needs to make systematic distinctions between the use of intensional and extensional definition of objects. (In simple terms an intensional definition defines *what* something is or does; an extensional definition refers to *how* it is constructed.)

- **Extensional** definition of objects is *a necessary characteristic of object-oriented programming languages.* Their inheritance hierarchy points at ancestral objects, the realizations of which are the shared resources to which further objects are bound (it defines *how* objects are constructed).

  (But intensional definition is inherent in ADTs, therefore object-oriented programming languages that have data abstraction also include intensional definition.)

- **Intensional** definition of objects is *a necessary characteristic of an object-oriented ODP specification language*. ODP-objects are defined in terms of what they do (i.e. the behaviour visible at their external interaction points, their interfaces). Their inheritance hierarchy defines units of behaviour specification (it defines *what* objects do).

  (Extensional definition of ODP-objects would be inconsistent with distributed processing, because practical considerations of scaling and performance mean that non-local resources cannot be shared like local ones.)

Essentially the same points are made by Peter Wegner in [1], although without making the extensional / intensional distinction explicit.

The same mathematical model of the inheritance hierarchy [15] works for both styles, intensional and extensional. It is also independent of whether the word "type-definition" or "class-definition" are used for the defining abstractions that are organized and inherited.

In ANSA we have chosen to associate the terms "class" "subclass" "class-definition" etc. with extensionally defined object structure, and the terms "type" "subtype" "type-definition" etc. with intensionally defined object structure. The choice of terminology is arbitrary (but is broadly consistent with the way the term "class" is used in Smalltalk, and "type" in ADTs).

The outcome is that we now have *"interface types"* defined by *"interface type definitions"*; we use the term *"subtyping"* when talking about inheritance hierarchies. We avoid using the terms "class" etc. in this context. We have no inhibitions about using the term "type" to refer to *data types*, *operation types*, *event types* etc.; but I am not sure if we would go so far as to define inheritance hierarchies for them.

This is a conscious departure from the normal object-oriented practice of associating the term "inheritance" exclusively with "class" and extensional definition [1]. Something has to be done about this terminological trap, because the concept of "inheritance" derived from "inheritance hierarchies" is universally applicable. Perhaps the proper way to make this distinction is to use (re-invent ?) different terms like "intensional inheritance" and "extensional inheritance".

Formal Methods. An important next step in the ISO ODP-RM work will be decisions on what formal specification notations to use. There is already agreement in principle to use formal methods; but always in conjunction with natural language description acceptable to a wide non-specialist audience.

The work already described here has laid some of the foundations. The basic concepts and constructs will probably be defined using elementary set theory, as has already been done for the inheritance concepts. The modelling of interactions as events opens the way to using process algebras to specify object behaviour. Most probably the ISO LOTOS language [16] will be used. This is based on CCS, and includes a data typing language (ACT-ONE) which could be used for ADT definition of objects. Our definition of "event" also allows formal composition and decomposition of events. Our foundation definition of "occurrence" ties into physics formalisms.

There is also a need for an Interface Definition Language (IDL) in which to express interface type definitions (and thereby to define objects). Interface definitions expressed in an IDL can be exploited as a means of declarative specification of protocol, synchronization, atomicity, etc. In the ECMA work we will probably continue to use the IDL defined in the ECMA RPC standard [8]. This IDL is a proper subset of the abstract data syntax and operation definition notations used in Open Systems Interconnection (OSI) standards. ANSA have a prototype IDL which is positioned much closer to the way programming languages work.

Object Engineering. In ANSA we are developing a formulation of object modelling for distributed systems which we call "object engineering". This provides a simple, but formal, graphical notation ("ball and stick diagrams" as illustrated in figure 2). It is underpinned by use of set theory.

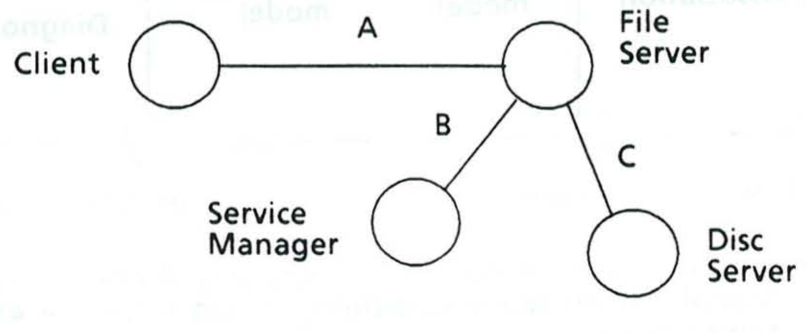


Figure 2: An Object Engineering diagram.

ANSA object engineering lays down general rules for the composition, decomposition and configuration of systems designs expressed as objects that interact with one another via various behaviour alphabets. It is mainly the work of Professor John Monk of the Open University, who is a part time member of the ANSA project team. I had hoped to present it more fully in this lecture, but have not yet had the time to master it myself in sufficient detail. For those who would like further details, preliminary documentation is available now from ANSA. Comprehensive and stable documentation should be available towards the end of the year, and is likely to be contributed to the next ISO meeting (December 1988).

Multi-interface Objects. In ANSA and ECMA we have the concept of objects with multiple interfaces. I mention this because it is controversial. The initial reaction of some people is that it is heresy (if only because familiar object-oriented programming languages like Smalltalk have no such concept). But multiple interfaces are an obvious necessity in the different field of systems modelling, as illustrated by the trivial example in figure 2, where one object has three interfaces (and different roles w.r.t. each). The interface characterized by alphabet A is used by the Client; it might have operations like "open file" and "read record". The interface characterized by alphabet B is used by the Service Manager; it might have operations like "switch off the service" and "change security policy". The interface characterized by alphabet C is used by the File Server itself, and might have operations like "read sector".

Figure 3 is a more complex example copied from an ANSA document. The details need not concern us, but it illustrates more of the range of object roles and granularity that are within the scope of ODP and object engineering.

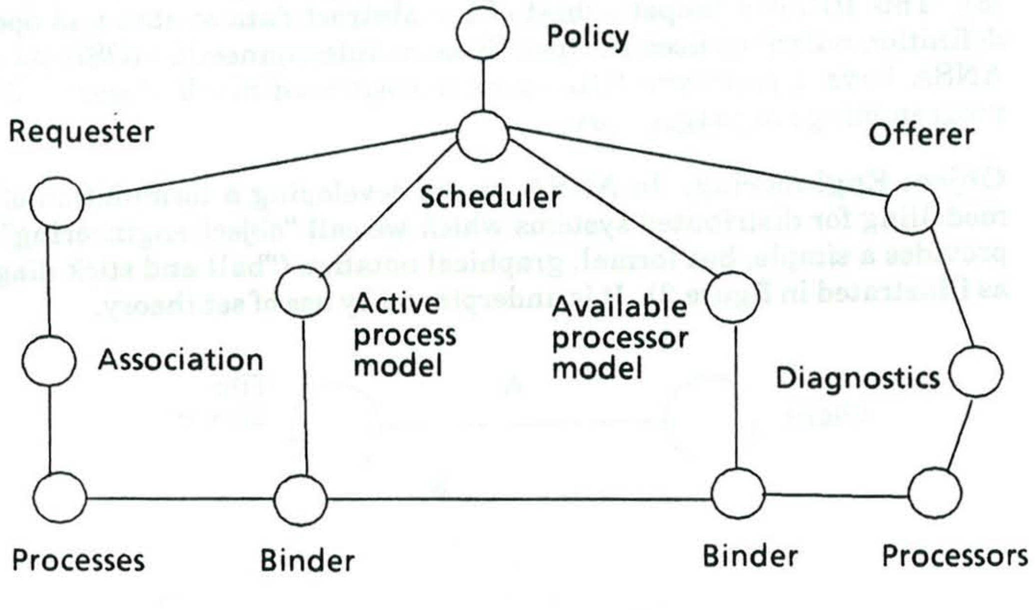


Figure 3: A more complex object engineering diagram (incomplete example of processor scheduling, taken from the ANSA object support system).

Trading. A major ingredient of the architecture developing in ANSA and ECMA is "trading", defined as the function of matching bids to use services ("imports") with offers to provide them ("exports"). These bids and offers are expressed primarily in terms of interface types. The configuration of distributed systems is defined in terms of "trading", "trading scopes" and related naming structures, access controls, etc. The ECMA ODP Support Environment [9] will define trading concepts and structure.

Object Groups. Concepts of object groups are important in distributed processing. An "object group" consists of multiple objects of which the externally visible collective behaviour is as if they were one object, with attendant simplifications. Acting in concert, the members of a group may achieve parallelism, resource sharing, resilience, fault tolerance, etc. There are related concepts of atomicity, synchronization, reliable broadcasts, etc. This leads into the world of fault-tolerant computing, which is why we maintain close links with Brian Randell and his team here at Newcastle. Such concepts will be included in the ODP Object Model.

9. Summing Up

I would like to summarize the main strengths and weaknesses of object-oriented concepts, as encountered so far in ODP standardization.

Strengths

- **Suitability.** There is absolutely no doubt that object-oriented techniques facilitate modular system design, and are the right approach for modelling distributed systems.

- **Expressive Power.** The evolving object model has appropriate expressive power for the specification of distributed processing systems. It is sufficiently abstract to avoid over-specification, and sufficiently formal to minimise the risk of specification errors and ambiguity.

- **Programming-in-the-large.** The further we go into distributed processing standardization, the more it becomes apparent that its main task is the integration of heterogeneous software components. It therefore has much in common with programming-in-the-large. Moreover, programming-in-the-large needs the distribution transparencies provided by distributed processing technology. The same kind of object-oriented abstractions are applicable to both these fields.

- **Implementation independence.** We have found that distributed systems structure can be defined in object-oriented terms, irrespective of whether the implementations use object-oriented programming languages or object-oriented operating systems. This de-coupling is vitally important, because most of today's programming languages and operating systems are not object-oriented.

- **Simplicity.** Although there is a persistent fog of confusions (see below), object-oriented concepts are a source of simplifications which greatly increase our ability to design, understand, build and operate complex distributed processing systems.

Weaknesses

- **Absence of General Theory.** There is, as yet, no generally accepted theory of objects in computing science. There is a corresponding lack of consistent definitions and consistent use of concepts and terminology.
- **D.I.Y.** Consequently, we are having to take a "do it yourself" (DIY) approach to object modelling in our field, with attendant risks.

The remaining weaknesses are essentially human problems arising from the above absence of general theory and related gaps in knowledge and insight. They could be classified as "confusions" rather than technical weaknesses:

- **Dogmatism.** Most individuals (present company excepted) tend to make passionate globally applicable assertions of the kind "Object-oriented is ...", depending on their own personal exposure to different facets of the subject. This leads to conflict which impedes consensus.
- **Concrete or abstract.** In natural language an "object" is any *thing* of interest ("this book", "that piece of computer software", etc.). But in programming languages, specification languages and modelling, an "object" is an *abstraction* of some such thing; it is not the thing itself. People have difficulty in maintaining these distinctions. At a more refined level they can be expressed as "extension" and "intension" (respectively). The overloading of subtle and often unnoticed differences of meaning onto the common term "object" is confusing.
- **Programming or specification.** Most programmers and system designers do not recognize distinctions between "object-oriented programming" and "object-oriented specification".
- **Types, Classes etc.** Most people who make standards get into muddles about classes, inheritance, types, ADTs, intension, extension, etc. (myself included).

These strengths and weaknesses provide fertile ground for the computing science curriculum and industrial training.

10. Conclusions

The practical value of "object-oriented" has long been recognized in the fields of programming languages and database. It is now time to recognize its worth as a principle for system organization in the twin fields of distributed processing and programming-in-the-large.

The great unknown for us in ODP/ANSA is: have we selected appropriate object-oriented concepts ?

It is also evident that the computer scientist who writes the definitive textbook on object-oriented techniques will be able to sell a lot of copies to practitioners like us, and to the future generations of graduates we expect to employ. I hope it will be one of you.

References

[1] WEGNER, P. "Dimensions of Object-Based Language Design". OOPSLA '87 Proceedings, ACM 0-89791-247-0/87/0010-0168.

[2] ANSA. "ANSA Reference Manual". Release 00.03, June 1987. ANSA, 24 Hills Road, Cambridge CB2 1JP, UK.

[3] HERBERT, A.J. "The Advanced Network Systems Architecture Project". ICL Tech J., Nov..... 1987.

[4] ISO. "Information Processing - Open Systems Interconnection - Basic Reference Model". International Standard 7498. International Standards Organisation, Geneva.

[5] ISO. "Press Release - Open Distributed Processing". March 1988. ISO/IEC JTC1/SC21/WG7 N001. (Available from National Standards bodies).

[6] ISO. "Proposed revised text for the New Work Item on the Basic Reference Model of Open Distributed Processing". ISO/IEC JTC1/SC21 N1889. (Available from National Standards bodies).

[7] ISO. "Report on Topic 1 - The Problem of Open Distributed Processing". March 1988. ISO/IEC JTC1/SC21 N2507. (Available from National Standards bodies).

[8] ECMA. "RPC Basic Remote Procedure Call Using OSI Remote Operations". ECMA Standard 127. December 1987. ECMA, 114 Rue du Rhone, CH-1204 GENEVA, Switzerland.

[9] ECMA. "Open Distributed Support Environment (ODP-SE)". Draft Technical Report. June 1988. ECMA, 114 Rue du Rhone, CH-1204 GENEVA, Switzerland.

[10] BRENNER, J.B. "Open Distributed Processing". ICL Tech. J., Nov... 1987.

[11] ISO. "Working Document on the Framework of Abstractions". June 1988. ISO/IEC JTC1/SC21/WG7 N021. (Available from National Standards bodies).

[12] SOWA J.F.; "Conceptual structures - information processing in mind and machine". Addison Wesley, USA (1984).

[13] ZACKMAN, J.A.; "A framework for information systems architecture". IBM Systems Journal Vol 26 No 3, 1987.

[14] ISO; "Modelling Techniques for the Specification of the ODP Reference Model". June 1988. ISO/IEC JTC1/SC21/WG7 N022. (Available from National Standards bodies).

[15] CUSACK, E.; "Fundamental Aspects of Object Oriented Specification". British Telecom Technical Journal, July 1988.

[16] ISO. "Information Processing - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour". International Standard 8807. International Standards Organisation, Geneva.

CONCLUSIONS

- **Time to recognize object oriented as a systems organization principle.**
- **In the twin fields of distributed processing and programming in the large.**
- **Have we selected appropriate object - oriented techniques ?**
- **Who will write the definitive textbook on object oriented techniques ?**

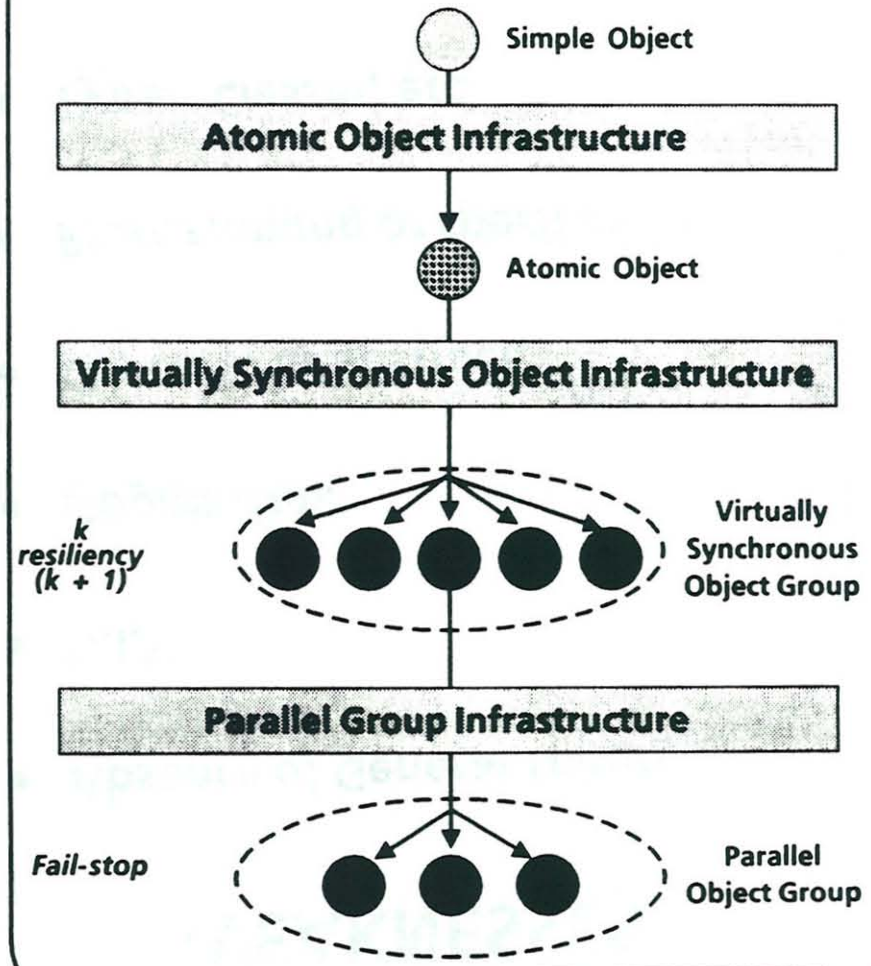
WEAKNESSES

- **Absence of General Theory**
- **D.I.Y.**
- ***Dogmatism.***
- ***Concrete or abstract.***
- ***Programming or specification.***
- ***Types, Classes, etc.***

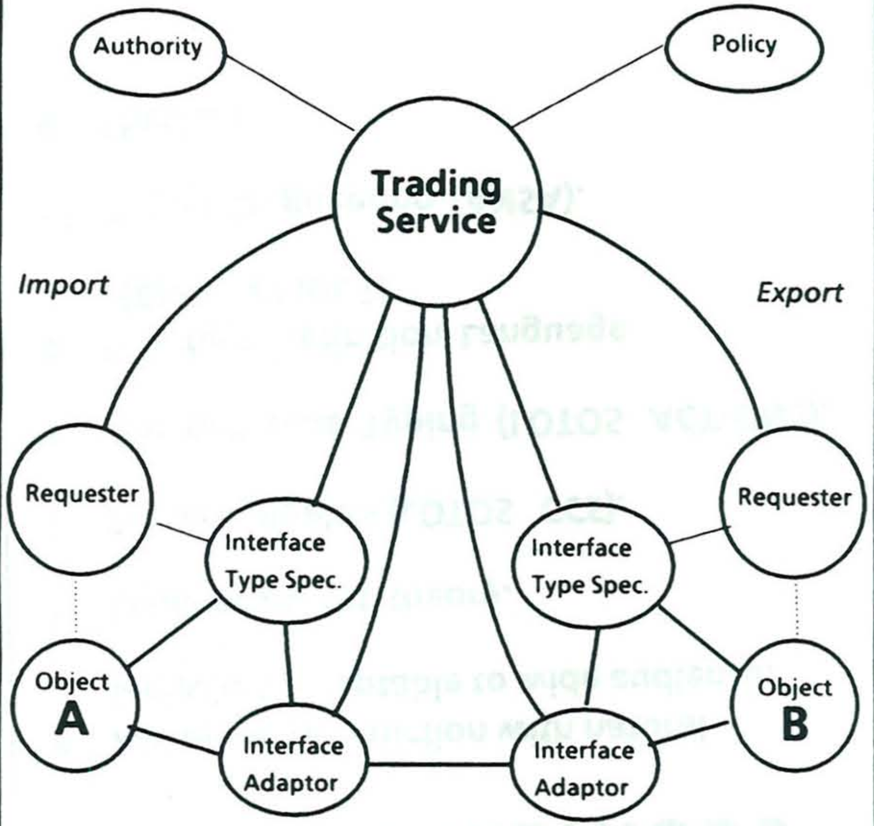
STRENGTHS

- Suitability
- Expressive Power
- Programming in the large
- Implementation independence
- Simplicity

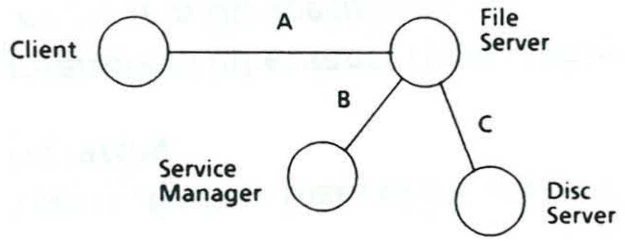
OBJECT GROUPS

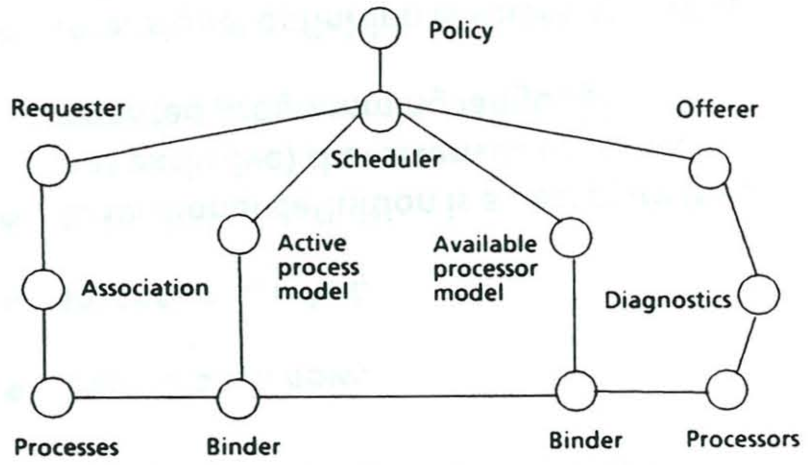

Simple Object
Atomic Object Infrastructure
Atomic Object
Virtually Synchronous Object Infrastructure
k resiliency (k + 1)
Virtually Synchronous Object Group
Parallel Group Infrastructure
Fail-stop
Parallel Object Group

TRADING



OBJECT ENGINEERING EXAMPLES





FORMAL METHODS

- Always in conjunction with natural language acceptable to wide audience.
- Elementary set theory.
- Process algebra (LOTOS *CCS*).
- Abstract Data Typing (LOTOS *ACT-ONE*).
- Interface Definition Language (ECMA 127 IDL ?).
- Object Engineering (ANSA).
- Physics !

INTENSION OR EXTENSION ?

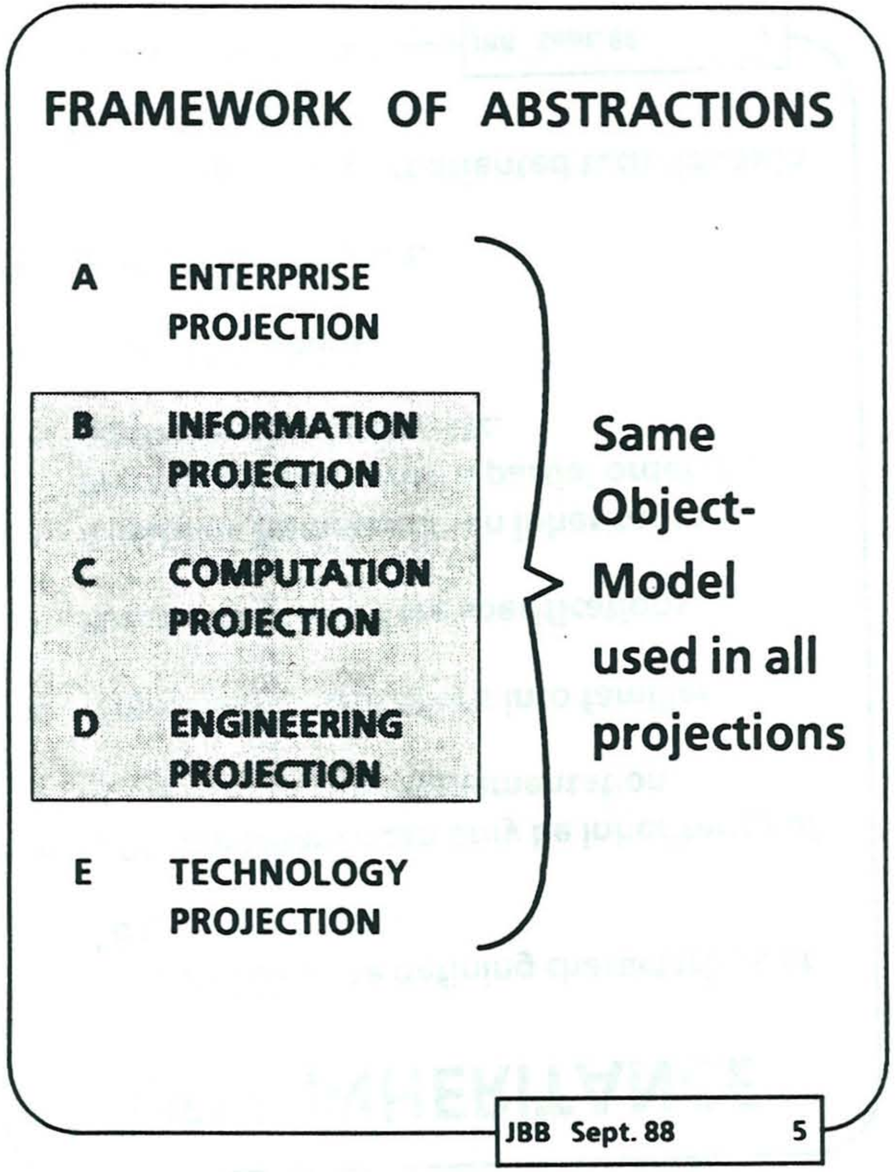
- Extension ... *how*.
- Intension ... *what*.
- Extensional definition is a necessary (but not exclusive) characteristic of object oriented programming languages.
- Intensional definition is a necessary (but not exclusive) characteristic of object oriented ODP specification.
- Two distinctive cases of inheritance hierarchy.
- ***Intensional inheritance*** *(type / subtype), inherent in OO specification languages ?*
- ***Extensional inheritance*** *(class / subclass), inherent in OO programming languages ?*

ODP INHERITANCE

- Inheritance is the defining characteristic of "object oriented".
- ODP inheritance can only be inheritance of specification, not implementation.
- Organizes specifications into families.
- Hierarchy of interface specifications.
- Algebraic framework: an inheritance hierarchy is a set with a partial order, an equivalence relation, etc.
- Strict inheritance.
- Multiple inheritance.
- Free choice of object oriented techniques in implementation.

ODP OBJECT MODEL REQUIREMENTS

- Specification Language (not programming language).
- ODP-object = an abstraction to model arbitrary components of distributed systems.
- Objects defined in terms of the interactions in which they can participate.
- Interactions defined in terms of shared events.
- Interface specifications are the defining abstraction for interactions.
- Specification families organized by inheritance hierarchy.



FRAMEWORK OF ABSTRACTIONS

A ENTERPRISE PROJECTION

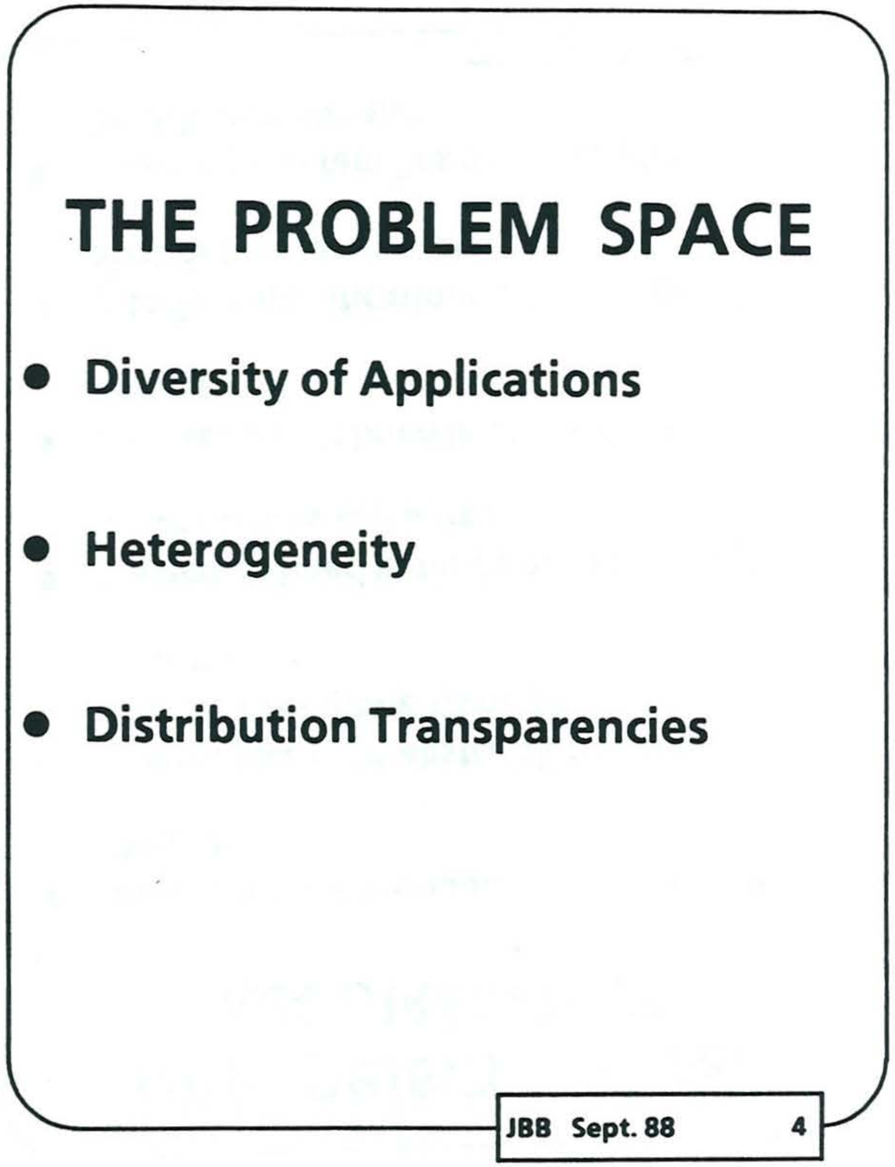
B INFORMATION PROJECTION

C COMPUTATION PROJECTION

D ENGINEERING PROJECTION

E TECHNOLOGY PROJECTION

Same Object-Model used in all projections



THE PROBLEM SPACE

- Diversity of Applications
- Heterogeneity
- Distribution Transparencies

ODP

- **Open** = standards to facilitate industrialization.

- **Distributed Processing** = computation via modular computing agents.

- **Open Distributed Processing** = standards to facilitate industrialization of computation via modular computing agents.

WHO WE ARE

- **ANSA:**

  UK Alvey Project;

  Advanced Network Systems Architecture;

  BT, DEC, GEC, HP, ICL, ITT Olivetti, Plessey, Racal.

- **ISO:**

  International Standards Organization.

- **ECMA:**

  European Computer Manufacturers Assn.

- **ICL:**

  Systems supplier;

  Integrating heterogeneous distributed systems.

Twenty-first University of Newcastle upon Tyne
International Seminar on Teaching of Computer Science
at University Level

OBJECTS AS A SYSTEM ORGANIZATION PRINCIPLE

John Brenner

ICL, Bracknell, UK

DISCUSSION

Professor Bayer raised the problem of the way in which multiple interfaces could be handled formally. In the discussion that followed Professor Nygaard suggested an introduction of the concept of internal objects to the framework presented by Mr. Brenner.

Professor Lee expressed an opinion that a possible reason for weaknesses of the presented approach might be a distinction which is being made between the behaviour of an object considered as an realisation of an abstract type, and the behaviour of the objects of that type in terms of specifications. Mr. Brenner replied that he would always like to think about objects as abstractions which only have specified interfaces, but due to some difficulties in the formal treatment it was necessary at some stages of the development to refer to the behaviour of an object as a realisation of an abstract type.

In the final part of the discussion Mr. Kerr and Mr. Brenner briefly discussed the relationship between the approach presented in the talk and some of the aspects of a research project which is currently carried out in the Computing Laboratory. The discussion was mainly focused on the differences and similarities between interface languages and (standard) languages for object-oriented systems.