SUPPORT ENVIRONMENT FOR OPEN DISTRIBUTED PROCESSING (SE-ODP)

ECMA TR/49

December 1989
SUPPORT ENVIRONMENT
FOR
OPEN DISTRIBUTED PROCESSING
(SE-ODP)

ECMA TR/49

December 1989
BRIEF HISTORY

This ECMA Technical Report was prepared by TC32-TG2.

Drafting of an ECMA Technical Report on an Open Distributed Application Support Environment (DASE) was started in mid 1986.

This work was temporarily suspended during 1987 while TG2 concentrated on contributing to the start-up phase of the new work item on the "Reference Model for Open Distributed Processing" (RM-ODP) in ISO, and produced the ECMA-127 OSI RPC Standard to help bridge between OSI and ODP.

Production of this TR resumed in mid 1988, by which time the outlines of ODP architectural structure had been agreed in ISO. The proposed support environment has been aligned to the RM-ODP structure, and renamed accordingly.

The final draft was completed in June, 1989.

Adopted as an ECMA Technical Report by the General Assembly of 14th December 1989
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECTION ONE - GENERAL</td>
<td>1</td>
</tr>
<tr>
<td>1. SCOPE</td>
<td>3</td>
</tr>
<tr>
<td>2. FIELD OF APPLICATION</td>
<td>3</td>
</tr>
<tr>
<td>3. CONFORMANCE</td>
<td>3</td>
</tr>
<tr>
<td>4. REFERENCES</td>
<td>3</td>
</tr>
<tr>
<td>5. DEFINITIONS</td>
<td>4</td>
</tr>
<tr>
<td>SECTION TWO - TECHNICAL INTRODUCTION</td>
<td>5</td>
</tr>
<tr>
<td>6. OVERVIEW OF TECHNICAL INTRODUCTION</td>
<td>7</td>
</tr>
<tr>
<td>7. WHAT IS AN ODP SUPPORT ENVIRONMENT?</td>
<td>7</td>
</tr>
<tr>
<td>8. CHARACTERISTICS OF DISTRIBUTED APPLICATIONS</td>
<td>9</td>
</tr>
<tr>
<td>SECTION THREE - DESCRIPTIVE CONCEPTS</td>
<td>15</td>
</tr>
<tr>
<td>9. OVERVIEW OF DESCRIPTIVE CONCEPTS</td>
<td>17</td>
</tr>
<tr>
<td>10. REFERENCE MODEL OF ODP</td>
<td>17</td>
</tr>
<tr>
<td>11. VIEWPOINTS</td>
<td>18</td>
</tr>
<tr>
<td>12. CLARIFICATION OF SCOPE</td>
<td>19</td>
</tr>
<tr>
<td>13. OBJECT MODELLING TECHNIQUES</td>
<td>21</td>
</tr>
<tr>
<td>SECTION FOUR - SE-ODP ARCHITECTURE SPECIFICATION</td>
<td>25</td>
</tr>
<tr>
<td>14. OVERVIEW OF SE-ODP ARCHITECTURE</td>
<td>27</td>
</tr>
<tr>
<td>15. DESCRIPTION IN THE COMPUTATION VIEWPOINT</td>
<td>27</td>
</tr>
<tr>
<td>16. DESCRIPTION IN THE ENGINEERING VIEWPOINT</td>
<td>29</td>
</tr>
<tr>
<td>17. DESCRIPTION IN THE TECHNOLOGY VIEWPOINT</td>
<td>32</td>
</tr>
<tr>
<td>18. SE-ODP OBJECT INTERACTIONS</td>
<td>33</td>
</tr>
<tr>
<td>19. SE-ODP PROCESSING MODEL</td>
<td>37</td>
</tr>
<tr>
<td>20. SE-ODP COMMUNICATIONS MODEL</td>
<td>39</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>21. APPLICATION CONFIGURATION</td>
<td>40</td>
</tr>
<tr>
<td>22. SE-ODP NAMING &amp; BINDING</td>
<td>43</td>
</tr>
<tr>
<td>23. SE-ODP DISTRIBUTION TRANSPARENCY TECHNIQUES</td>
<td>46</td>
</tr>
<tr>
<td>24. SE-ODP RUNTIME SUPPORT</td>
<td>52</td>
</tr>
<tr>
<td>25. SE-ODP MANAGEMENT ASPECT</td>
<td>57</td>
</tr>
<tr>
<td>26. SE-ODP SECURITY ASPECT</td>
<td>58</td>
</tr>
<tr>
<td>SECTION FIVE - PROPOSED STANDARDIZATION</td>
<td>61</td>
</tr>
<tr>
<td>27. OVERVIEW OF PROPOSED STANDARDIZATION</td>
<td>63</td>
</tr>
<tr>
<td>28. SE-ODP CONFIGURATION STANDARD</td>
<td>64</td>
</tr>
<tr>
<td>29. SE-ODP INTERACTION SEMANTICS STANDARD</td>
<td>65</td>
</tr>
<tr>
<td>30. SE-ODP INTERFACE DEFINITION LANGUAGES STANDARD</td>
<td>66</td>
</tr>
<tr>
<td>31. SE-ODP INTERCONNECTION STANDARD</td>
<td>67</td>
</tr>
<tr>
<td>32. SE-ODP INFRASTRUCTURE INTERFACES STANDARD</td>
<td>68</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>71</td>
</tr>
<tr>
<td>APPENDIX A - BIBLIOGRAPHY</td>
<td>73</td>
</tr>
<tr>
<td>APPENDIX B - BASIC CONCEPTS</td>
<td>79</td>
</tr>
<tr>
<td>APPENDIX C - DISTRIBUTED SYSTEM CONCEPTS</td>
<td>95</td>
</tr>
<tr>
<td>APPENDIX D - TRADING CONCEPTS</td>
<td>97</td>
</tr>
<tr>
<td>APPENDIX E - INDEX OF TERMINOLOGY</td>
<td>105</td>
</tr>
</tbody>
</table>
SECTION ONE - GENERAL
1. SCOPE

This ECMA Technical Report results from a preliminary study of architecture to provide a standard Support Environment for Open Distributed Processing (SE-ODP). It is primarily concerned with distributed processing in terms of the structure of applications software, and how to support this.

This ECMA Technical Report:

(a) explains what the SE-ODP is about (see Section Two);
(b) defines concepts with which to describe SE-ODP architecture (see Section Three);
(c) specifies SE-ODP architecture (see Section Four);
(d) identifies proposed areas of standardization (see Section Five);
(e) provides supporting tutorial and background material (see Appendices).

The architecture specified in (c) is not complete or definitive. It is only intended to be sufficient to provide an initial basis for the further SE-ODP standardization work identified in (d).

2. FIELD OF APPLICATION

SE-ODP standardization applies to distributed processing that may be implemented via the products of multiple independent suppliers.

3. CONFORMANCE

Conformance requirements are outside the scope of this ECMA Technical Report, and are a matter for the SE-ODP standards proposed in Section Five.

It is intended that this ECMA Technical Report itself complies with the requirements of the ISO Reference Model of Open Distributed Processing (RM-ODP); see clause 10.3.

4. REFERENCES

ISO 8824  Information Processing Systems - Open systems
        Interconnection - Specification of Abstract Syntax Notation
        One (ASN.1)

ISO 8825  Information Processing Systems - Open systems
        Interconnection - Specification of Basic Encoding Rules One
        (BER.1)

ISO 9072/1 Remote Operations Part 1: Model, Notation and Service
       Definition.

ISO 10026  Information Processing Systems - Open Systems
        Interconnection - Distributed Transaction Processing.

CCITT Rec. X.500  The Directory.

series

References to documents that are not Standards publications are provided in the
Bibliography in Appendix A. They are referenced in the body of the Technical
Report by names and dates in square brackets [ <name> yy].

5. DEFINITIONS

For the purposes of this Technical Report the following definitions apply:

processing: programmed activity executed by computers.

distributed processing: processing that may span separate computer address spaces.

Reference Model of Open Distributed Processing (RM-ODP): a reference model to
be defined by ISO/IEC JTC1 SC21 WG7.

open distributed processing: distributed processing conforming to requirements
specified in the RM-ODP.

Support Environment for Open Distributed Processing (SE-ODP): technical
provisions to support open distributed processing.

Definitions of other terms are included in definition subclauses within the
individual clauses.

Some definitions are repeated in clauses where they are particularly relevant.

Appendix E is an alphabetical index of all the defined terms and abbreviations
used in this Technical Report. It provides references to where each is defined.

As an aid to understanding definition structure and use, references to defined
terms are usually in italics.
6. **OVERVIEW OF TECHNICAL INTRODUCTION**

The clauses in Section Two provide a technical introduction to the subject area of the Technical Report.

Clause 7 describes what a support environment for *open distributed processing* is required to do.

Clause 8 describes application characteristics that the *SE-ODP* should support.

7. **WHAT IS AN ODP SUPPORT ENVIRONMENT?**

7.1 **General**

This clause describes, in general terms, what a support environment for *open distributed processing* is required to do.

The *SE-ODP* is concerned with an infrastructure for *distributed processing* that goes beyond interconnection and data-communications.

*NOTE 1: The terminology relating to system concepts which is used here is defined in Appendix B.*

7.2 **Modular Software and Distributed Processing**

There is a fundamental relationship between modular software and *distributed processing*. This relationship is described below.

Software engineering practice favours the specification and design of applications by means of program modules with defined interfaces. Behind these interfaces, the contents of each module are encapsulated and hidden from other modules. All *interactions* between such modules are formulated as *interactions* via these interfaces, and not via shared data. See Figure 1.

![Figure 1 - Modular Software](image)

This notion of modules is particularly well-adapted to designing *distributed applications*, in which the program modules comprising the application may be in different locations in a *distributed IT system*. The *services* of such a *distributed application* would then be provided by co-operation between the modules. Some of the modules may be sited at physically separated computers, while others may be co-located on the same computer.

The module interfaces, and the *interactions* via those interfaces, should be essentially the same wherever the modules are located. This facilitates resource sharing and supports the re-usability and portability of the program modules.

The software modularity is only completely general if it allows modules to be constructed independently, and to be executed autonomously in heterogeneous
systems. This allows use of diverse programming languages, computers, operating systems and interconnection networks. Further requirements for full generality also include: federated control of the systems concerned (i.e. no one management authority), and assurance of particular quality attributes such as dependability (see 8.9.1), and scaling (i.e. adaptability for small and very large aggregations of modules).

From the above, the key ingredients can be summarized as:

(a) modules + interfaces + encapsulation (this is distinctive of modular software);
(b) autonomy + separation (this is distinctive of distributed processing);
(c) dependability + scaling (this is a practical necessity);
(d) portability (for re-use and re-location of software);
(e) heterogeneity + federation (the extra needs that are distinctive of open distributed processing).

We are concerned here with the whole combination (a), (b), (c), (d), (e), and place particular emphasis on issues arising from heterogeneity, distributed ownership, distributed authority, and distributed operational control.

7.3 Support Environment

Some kind of support environment, or infrastructure, is required to achieve such distributed processing.

The Support Environment for Open Distributed Processing (SE-ODP) is focussed on the application programmer’s view of a distributed system. This view is illustrated informally in Figure 2, in which the modules (circles) that are linked together are perceived to execute within some all-pervading support environment (the rectangle). The SE-ODP is an environment for interaction and binding between software modules (located in physically separate machines, or co-located). It facilitates the construction, operation and maintenance of distributed application software systems.

![Figure 2 - Support Environment](image)

The support environment for distributed processing that is the subject of SE-ODP standardization is intended to be applicable where there is heterogeneity
and multi-vendor procurement, with consequent requirements for open standards.

The SE-ODP may be viewed as a means to provide "distributed object access", and nothing more (see note). If provided in a sufficiently general way, this one function of "distributed object access" is a sufficient basis for organizing provision of arbitrary function by modular software using it. Therefore, the SE-ODP is an enabler for distributed processing in general, and need itself have no other function.

NOTE 2:
This is a different role from that of the operating systems with rich functionality which manage computer resources etc. (and are used by the SE-ODP provisions); e.g. Posix.

7.4 SE-ODP Prototypes

The SE-ODP proposals in this ECMA Technical Report are based on practical experience of several ECMA Member Companies with prototype implementations of support environments for heterogeneous distributed processing. In particular the following are sources of input to this ECMA work.

(a) The ANSA Testbench implementation by the UK Alvey ANSA project, now the Esprit ISA project. Members of this Esprit II consortium are BT, Digital, GEC/GPT, Ellemtel, HP, ICL, ITL, Patras University, Philips, Siemens, SEPT and STC. Comprehensive documentation of its current architecture and software is provided in [ANSA 89].

(b) The DACNOS implementation by the IBM European Networking Centre (Heidelberg) and the University of Karlsruhe [KRUEGER 88]. This is a prototype network operating system for multi-vendor environments.

(c) The implementation of the DELTASE prototype by the ESPRIT Delta-4 project [DELT A 4, 88]. This is concerned with fault-tolerant open system structure, intended primarily for process-control applications. Members of the original Esprit I Consortium were Bull, BASF, Ferranti CSL, GMD First, IEI-CNR, Fraunhofer Inst., INESC, Jeumont-Schneider, LAAS-CNRS, LGI/EMAG, MARI, Teletra, University of Bologna.

Development of these and other SE-ODP prototypes continues, and will be reflected in further ECMA work on this subject.

8. CHARACTERISTICS OF DISTRIBUTED APPLICATIONS

8.1 General

This clause describes application characteristics that the SE-ODP should support. The description is in terms of distributed applications and application components. The latter is a more general way of viewing the "program modules" of clause 7.

8.2 Definitions

The following definitions apply.

computer application: productive activity exerted via computers.
application program: the autonomously executable specification of (a part of) a computer application.

application component: an application program considered in terms of its contribution to the composition of some computer application whole.

distributed application: a computer application composed of discrete application components.

8.3 Application structure

No distinction is made here between "user" applications (e.g. order entry or process control), and those applications that might be classified as part of the "system" (e.g. editing, filing, printing, database, directory). Any computer application is a potential candidate for open distributed processing, and hence use of the SE-ODP.

The granularity of application components depends on a number of factors, including choices made by the designer. Typically, an application component would be a single unit of compilation (and would correspond to a software module as described in 7.2). The level of application component granularity of primary interest in the SE-ODP standardization is that at which the system designer wants distribution transparency (see 8.6).

8.4 Interaction

An interaction between application components is considered to be external and computational: each application component is wholly external to the other, and interaction is defined in terms of computational semantics (synchronized flow of program control and data). At the programming level, invocation of these interactions is expressed using language constructs such as procedure calls (hence "remote procedure calls").

A simple model of such an interaction is illustrated in Figure 3. The interaction is labelled with a symbol "X" that is intended to identify its specification.

![Figure 3 - Interaction between separate components](image)

This requirement to define interactions in terms of computational language abstractions is different from conventional networking interconnection. In networking the interactions are usually defined in terms of application protocols and their message structure, together with interfaces and finite state machines via which the protocols are mechanized. Abstracting away from such mechanism details is a characteristic of distributed processing (e.g. ODP) as considered here. ODP protocols would normally be generated by automated refinement from the stylized computational specifications of interactions.
8.5 Configurations

Interaction occurs in the context of a binding; i.e. the configuration of components of a distributed application is defined by the bindings between them. These bindings may be static bindings (persistent, established prior to run-time) or dynamic bindings (temporary, established at run-time).

Figure 4 illustrates some examples of such external bindings and interactions. Example 1 is like that already illustrated in Figure 3. Example 2 is the same, except that the interaction is different (interaction A in the first example, B in the second). In example 3 there are several different interactions between the two components, including multiple interactions of the same kind. In example 4 there is a mesh of various interactions between several components.

8.6 Distribution Transparency

Distribution transparency is concerned with hiding the effects of distribution (and is explained more fully in B.4).

Application components usually require a high degree of distribution transparency; e.g. complete hiding of the location (and re-location) of the application components with which they interact.

However, certain application components may require visibility of component location, component failure, etc., in order to manipulate to advantage the effects of separation (e.g. to achieve resilience, parallelism, reconfiguration).

The degree of distribution transparency should therefore be selectable by application designers and by the programs that use the SE-ODP. But incomplete distribution transparency may impact software portability (see 8.7).
In practice, the effects of distribution cannot always be completely hidden; for example, the differences in response times due to transmission and scheduling delays, or the speed-up due to concurrent execution on separate computers.

8.7 Software Portability

The software of distributed applications (or individual application components), may need to be portable across different operating systems, computers and networks. This is only considered here in terms of portability of application design and program source code (not portability of executable binary code).

A basic level of portability is inherent in the potential for re-location that is provided by distribution transparency, as follows.

Any component (A) that accesses services external to it (x, y, z ...) in a way that preserves distribution transparency, should be able to continue to access these services, irrespective where it (A) is located in the distributed system. Access is also independent of where the services (x, y, z ...) are each provided from (locally or remotely), and independent of what components (B, C, D ...) provide these services (x, y, z ...), and how they (B, C, D ...) are constructed. This is, of course, subject to practical constraints on connectivity, ownership, security, response times, etc.

This potential for re-location can be exploited to achieve general portability, as follows:

If a component (A) only obtains services external to it (x, y, z ...) via the SE-ODP (and not in ways dependent on its local environment), then it (A) is logically independent of its local environment and is portable to anywhere the services concerned (x, y, z, ...) are accessible via the SE-ODP.

It is assumed that program modules are written in languages for which appropriate language interpreters (e.g. compilers) are generally available.

8.8 Preservation of Existing Investment

The SE-ODP should allow continued successful use of investment in existing programming languages, compilers, language libraries, operating systems, databases, networks, terminals and access interfaces. However, the SE-ODP may also open up opportunities to exploit new languages and new facilities.

It is important that the SE-ODP is such that existing human skills (particularly those of application system users, analysts, designers and programmers) can be used productively for the construction and integration of distributed applications, and with a minimum of re-training.

8.9 Special Requirements

Some special requirements are identified in this subclause.

8.9.1 Dependability

Dependability concepts and related quality attributes are defined in [Laprie 85].
The concept of dependability includes reliability, availability, performance, security and safety. Many applications using the SE-ODP would have stringent requirements for dependability.

8.9.2 Real Time Applications

The SE-ODP field of application includes real-time applications (e.g. process control).

Real-time applications require assurance concerning the timeliness of service provision. This may range from a firm guarantee of service provision within a defined deadline, to a high expectation of service provision within a defined elapsed time. The timeframe may range from milliseconds to hours. In many cases, real-time applications also require a high degree of dependability (see 8.9.1).

All distributed applications are real-time applications, in that they require some assurance of timeliness of interactions; but for many applications using the SE-ODP the real-time requirements are not stringent.

8.9.3 Clock Synchronization

The synchronization of clocks in physically separate locations is a major issue in the design and implementation of distributed systems.

The granularity of time measurement, the accuracy of clock synchronization, and the variability of end-to-end interaction delays are inter-related matters.

Operation of the SE-ODP infrastructure itself depends on (loosely) synchronized clocks. Their granularity and accuracy should be appropriate to the applications concerned. The clock values should also be accessible for applications to use for their own purposes.

8.9.4 Isochronous Interactions

There are many potential computer applications which have isochronous interactions between their components (e.g. voice and image interactions in real-time).

However, most existing computer applications involve only anisochronous interactions; e.g. current OSI standards for the upper layers only support anisochronous interactions (not isochronous).

Isochronous interactions between software modules raise difficult technical issues concerning software design and programming language structure.

8.9.5 Multi-media

Support for multi-media interactions is also required (e.g. data + voice + image). This includes problems of how to synchronize concurrent voice, image and data interactions between multiple endpoints, with transmission via channels with different delay characteristics.

8.9.6 Computer Aided Software Engineering

Computer Aided Software Engineering (CASE) is of increasing importance in the industry, and is the subject of active standardization work in ECMA (TC33), the IEEE, ISO, and many other bodies.
Distributed applications using the SE-ODP should be able to benefit from CASE technology. Likewise, the distributed processing technology from SE-ODP standardization should be applicable to CASE systems (where the latter are constructed as heterogeneous distributed applications).

Therefore, the design of the SE-ODP is required to facilitate use of CASE techniques. A particular characteristic should be ability to use the design databases on which application development systems are increasingly centred. These database systems are variously referred to as "data dictionaries", "data repositories", and "information resource directory systems (IRDS)".

These CASE concerns are mainly addressed in the proposed SE-ODP standardization identified in clause 28.
SECTION THREE - DESCRIPTIVE CONCEPTS
9. OVERVIEW OF DESCRIPTIVE CONCEPTS
The clauses in Section Three set out a basic toolkit of concepts that are used to
describe and explain the SE-ODP and its use. The concepts and approach are
drawn from the work on the Reference Model of Open Distributed Processing (RM-
ODP) developing in ISO. The structure of the section is set out below.
Clause 10 outlines the work in ISO on the RM-ODP, and relates the SE-ODP to it.
Clause 11 introduces the viewpoints which are the main descriptive framework
identified in the RM-ODP.
Clause 12 describes differences of scope between ECMA SE-ODP standardization
and the ISO RM-ODP work item.
Clause 13 describes the way in which object model techniques are used in this

10. REFERENCE MODEL OF ODP

10.1 Specification
ISO is investigating and documenting a series of Topics that explore the subject
matter of the RM-ODP work item. At the time of preparing this ECMA
Technical Report there are ISO working papers documenting about half of these
topics; but there is as yet no ISO draft of the RM-ODP. ECMA participates
actively in this ISO work.

Material which is used in this Technical Report and should originate from the
RM-ODP is identified in 10.2 and is documented separately in Appendix B. This
is intended to reflect the current and expected results of the ISO architecture
work, and is therefore a prediction of RM-ODP content.

Appendix C considers different kinds of approaches to distributed systems, and
introduces the terms networking, network operating system and distributed
operating system. These distinctions may be needed in the RM-ODP.

NOTE 3:
Familiarity with the RM-ODP concepts and terminology is assumed in the remaining clauses of this
Technical Report. Therefore, Appendix B should be consulted at this point. If it were not separate
for the procedural reasons stated above, the material in Appendix B would be organized as part of
this Section of this TR.

10.2 Usage of the RM-ODP
This subclause identifies the main items from the RM-ODP on which the SE-
ODP depends. (There is likely to be much else in the RM-ODP which does not
directly affect the SE-ODP and therefore need not be considered here.)
The RM-ODP (via Appendix B) is the source of the following ingredients of the
SE-ODP architectural structure.

(a) It provides terminology and concepts that are general to all open
distributed processing (see Appendix B).
(b) It identifies **distribution transparency** as the distinctive architectural characteristic inherent in **distributed processing**. It defines several different kinds of **distribution transparency** which are needed in various degrees, depending on circumstances.

(c) It distinguishes between **descriptive** techniques applicable to all **distributed processing**, of whatever diversity, and the **prescriptive** formulation of what is specifically required for **distributed processing** to qualify as being **open**.

(d) It identifies **heterogeneity** as being inherent in **open distributed processing**.

(e) It will define an **object model** which is applicable to all modelling of **distributed processing** systems. This unification of modelling technique is essential to the architectural coherence of **ODP**.

(f) It defines **five viewpoints** via which different facets of a **distributed processing system** can be modelled separately and related together in a complete and systematic way.

The **SE-ODP** includes further levels of architectural detail, some of which may also be candidates for inclusion in the **RM-ODP**. Particular items in the latter category are the concepts of a **semantic model of interaction**, **interface trading**, relationships to **security architecture**, and identification of further **ODP standardization requirements**.

### 10.3 Conformance with the RM-ODP

At the time of preparing this Technical Report, ISO have not yet defined **RM-ODP** conformance requirements. The conformance clause of the **RM-ODP** is expected to define:

(a) requirements for **ODP standards** to use **RM-ODP** concepts and terminology, and to align to certain **ODP** reference points at which conformance is necessary to achieve **open distributed processing**; and

(b) requirements for **products** to conform to certain sets of standards in order to qualify as **ODP-conforming**.

The intended conformance by this Technical Report is (a); but strict conformance is not possible until after the **RM-ODP conformance clause** has been drafted.

### 11. VIEWPOINTS

#### 11.1 Introduction

**ISO ODP** has partitioned the problem space of **distributed processing** into a framework of abstractions, in which the primary structure is the five **viewpoints** referred to in 10.2 (f).

This partitioning is fundamental to the structure of the **SE-ODP**, and is briefly summarized here for the convenience of readers. A more complete description is given in Appendix B.5.
The whole information system is represented in each viewpoint, but with the emphasis on a particular concern. Collectively, the models in the five viewpoints are intended to model the main concerns in the design of distributed IT systems.

11.2 The five Viewpoints
The following viewpoints have been identified:

Enterprise viewpoint: this describes the information system in terms of what it is required to do for the enterprise concerned.

Information viewpoint: this concentrates on information structure, and the information flow of information systems. The rules and constraints that govern the manipulation of information are identified.

Computation viewpoint: this describes the computational characteristics of the information system, and the processes which change the information.

Engineering viewpoint: this describes the information system in terms of the engineering necessary to support the distributed nature of the processing.

Technology viewpoint: this concentrates on the technical artifacts from which the distributed processing system is built. E.g. it models the hardware and software that comprise the local operating systems, the input/output devices, storage and communications.

11.3 Implementation
These viewpoints distinctions are a modelling tool. They are not intended to imply any kind of layered implementation or any particular design process.

12. CLARIFICATION OF SCOPE

12.1 Introduction
This clause describes differences of scope between this ECMA SE-ODP standardization and the ISO RM-ODP work item (and ODP standardization in general).

12.2 Level of Detail
The SE-ODP standardization is concerned with more detailed and more localized standardization within the general structure (to be) provided by the RM-ODP.

This is analogous with the way in which OSI standards have been developed within the general structure provided by the OSI Reference Model, ISO 7498. (Another parallel is that some more detailed OSI work items started in ECMA and ISO before completion of the OSI Reference Model.)

12.3 Viewpoints
The SE-ODP standardization is concerned with all of the viewpoints (although to varying degrees). In the RM-ODP, data modelling in the information
viewpoint is likely to be of major importance (but is outside the SE-ODP scope). The RM-ODP standardization is also likely to explore further structure defined in terms of "aspects".

12.4 Subject Matter

The SE-ODP standardization is primarily concerned with distributed processing in terms of matters arising from distribution and the structure of software.

The RM-ODP standardization is apparently concerned with the structure of information systems more generally, and with providing a general framework in which to position many different kinds of standards (e.g. database standards, graphics standards, management standards). SE-ODP standardization does not attempt to do this.

12.5 Field of Application

The SE-ODP standardization is only concerned with information systems in which there is open distributed processing and use of ODP standards.

The RM-ODP standardization is also concerned with modelling any information system to which ODP standards might be applicable (i.e. it is intended to provide a "general descriptive model of all distributed processing", and not just a "prescriptive model of open distributed processing").

12.6 Focus

The SE-ODP standardization concentrates the core area of techniques for using and providing distribution transparency. This is only a subset of the whole problem space addressed by ODP standardization, but it is never-the-less large and complex.

The RM-ODP standardization, and ODP standardization more generally, has a yet larger and more diverse and more complex content.

12.7 Summary

These distinctions are summarized in the Venn diagram in figure 5 (which is a notation not intended to be to scale).

Figure 5 - Nested scopes
13. OBJECT MODELLING TECHNIQUES

13.1 Introduction

This clause describes the way in which the general object model defined in clause B.6 is used to describe information systems and the support structure provided by the SE-ODP.

Recipes for arrangement and transformation of objects are provided here for use in the computation viewpoint, the engineering viewpoint and the technology viewpoint. These recipes are only described to a level sufficient for the immediate purpose (and some important details are omitted). More complete and rigorous specification is for future study.

NOTE 4:
Clause B.6 of Appendix B should be read at this point, because it provides the base definitions and concepts used here.

13.2 Definitions

The following definitions apply.

processing component: any object that is considered to contribute to processing.

interface object: an object which is considered to be the locus for interaction.

abstract interface: an interface object at which interaction is defined without reference to representation.

ODP interface: an abstract interface defined in ways prescribed by SE-ODP standardization (see note).

ODP component: a processing component defined in terms of one or more ODP interfaces with which it has a connexion.

native component: a processing component that is not constrained by SE-ODP standardization (see note).

final form model: the object diagram(s) providing complete description of the information system considered, as visible in a particular viewpoint.

NOTE 5:
The definitions of "ODP interface" and "native component" refer to "SE-ODP standardization", not "ODP standardization". The latter might include standardization of objects considered here to be "native components". When ODP standardization is at a more advanced stage of development, these particular definitions may need to be be reformulated.

13.3 Processing Components

ODP is essentially about processing and interaction between processing components. This is particularly the case in the computation viewpoint and engineering viewpoint, with which the SE-ODP is mostly concerned.

Therefore, in this clause the SE-ODP is considered in terms of processing components and interactions between them; joint actions other than interactions are not considered.
Most of the processing components considered are modelled primarily as ODP components. All other processing components are modelled as native components.

13.4 ODP Interfaces

By definition, a connexion between ODP components is always via an ODP interface.

The specification of an ODP interface defines all valid interactions that can occur at the interface object. The particular formulation of ODP interfaces (see clause 19) ensures that comprehensive specification of distribution transparency constraints can be a characteristic of all ODP interfaces.

Therefore, explicit specification of interactions and the provision of distribution transparencies are inherent in every connexion between ODP components. That is why the SE-ODP generally requires processing components to be modelled in this way.

An object diagram need only show an ODP interface where doing so is appropriate to the level of detail being modelled. For example, the two object diagrams in figure 6 are equivalent, because in both cases the interaction between the ODP components is the same (x); each diagram is a transformation of the other.

![Figure 6 - Equivalent object diagrams](image)

Figure 7 - Object diagrams each equivalent to both those in figure 6
This approach to interfaces is a progression from the formulation in [ANSA, 89]. Interfaces were previously modelled as being integral to the junction between an object and a connexion (and were thereby part of the basic formulation of the object model). In this Technical Report, interfaces are modelled as "first class citizens" which have names and attributes, and can exist separately in object diagrams. The main reasons for doing this are as follows.

(a) To simplify the formulation of the Basic Object Model in B.6, and to make it more generally applicable, interfaces are now a refinement added for use in particular viewpoints.

(b) To achieve a more comfortable fit to the notion that ODP components bind to ODP interfaces, and not directly to one another. This indirection is fundamental to the trading of interfaces (see Appendix D).

These formulations are equivalent, in that they are different ways of presenting the same information.

There is also a concept of "interface objects" in DACNOS [KRUEGER 88].

NOTE 6:
The concept of an "ODP interface" described in this clause should not be confused with the different (but related) concept of an "interface adaptor", described in clause 21 and used in 16.2.

13.5 Transparent Insertion of Objects

An ODP component which allows the interaction considered, and conforms to the same ODP interface specification, may be inserted transparently into a connexion. (Detailed formulation of this is for further study.)

The two equivalent object diagrams in figure 8 are an example. In the second diagram an ODP component is inserted between the ODP components shown in the first. (The transformations are essentially the same as those in figure 6.)

![Diagram showing transparent insertion of an object](image)

**Figure 8 - Insertion of an extra object**

Transformations of this kind may be applied to the resultant object diagram, again and again.

This recipe is particularly important in the engineering viewpoint, where it is used to insert ODP components that support distribution transparencies.
13.6 Native Components

Most computers, networks, software systems, etc. are (and will continue to be) defined outside SE-ODP standardization. They are whatever they are.

The SE-ODP therefore has to operate in the context of whatever is native to the world that exists independently of it. The concept of native components is used to model this real-world diversity and heterogeneity.

ODP components may have connexions to native components. A native component may have connexions to ODP components. These connexions are identified with interactions that are specific to the native component; i.e. the ODP component has to adapt to this heterogeneity.

The object diagram in figure 9 is an example in which an ODP component, defined in terms of a connexion (labelled x) to an ODP interface, also has a connexion to a native component. By definition, the specification of the interaction (y) with the native component does not originate from SE-ODP standardization (but SE-ODP standards might influence choices of which specifications to use).

![Diagram showing ODP interface, ODP component, and native component with connexions]

Figure 9 - How native components are modelled and hidden

The ODP component in figure 9 is in a position to hide the native component. Neither the existence of the native component, nor the interactions y, need be visible to other ODP components that participate in interactions x. The native component is then encapsulated behind an ODP interface by the ODP component. Any visible activity has been normalized.

This recipe for ODP components encapsulating the non-ODP world is fundamental to the operation of the SE-ODP, and is used for description in each of the viewpoints considered. It is at the heart of the technology viewpoint.

13.7 Object Diagrams

In each viewpoint the information system considered is modelled by means of object diagrams.

Visibility criteria particular to the viewpoint determine what kind of object is visible.

In any viewpoint there may be object diagrams at different levels of detail, and separate object diagrams for separate subsystems. The concept of a final form model brings all this together for each viewpoint.
SECTION FOUR - SE-ODP ARCHITECTURE SPECIFICATION
14. OVERVIEW OF SE-ODP ARCHITECTURE

The clauses in the previous Sections have introduced the subject area of the Technical Report, and have presented descriptive concepts. From this starting point, the clauses in Section Four specify the SE-ODP architecture.

The specification is neither complete nor definitive. It is only intended to be sufficient to provide an initial basis for the further SE-ODP standardization work identified in the clauses of Section Five. This further work, together with RM-ODP standardization in ISO, would provide the basis for definitive architectural specification, which might be the subject of a future ECMA SE-ODP standard.

Clause 15 describes and explains what is visible in object diagrams that model an information system in the computation viewpoint.

Clause 16 describes and explains what is visible in object diagrams that model an information system in the engineering viewpoint.

Clause 17 describes and explains what is visible in object diagrams that model an information system in the technology viewpoint.

Clause 18 describes the structure of interactions via ODP interfaces. This structure is such that interactions can be completely defined without reference to the way in which they are represented.

Clause 19 describes the SE-ODP processing model which relates together processing, memory and communications.

Clause 20 describes the Communications Model used for all interactions between processing components that are executed in separate address spaces.

Clause 21 describes concepts for the configuration of distributed information systems, and applies these concepts to some of the examples given in clauses 15, 16 and 17.

Clause 22 describes naming and binding techniques used with SE-ODP architecture.

Clause 23 describes the distribution transparency techniques used in SE-ODP architecture.

Clause 24 describes the SE-ODP in terms of its run-time support structure (termed SE-ODP runtime).

Clause 25 describes the management aspect of SE-ODP architecture.

Clause 26 describes the security aspect of SE-ODP architecture.

15. DESCRIPTION IN THE COMPUTATION VIEWPOINT

15.1 Introduction

This clause describes and explains what is visible in object diagrams that model an information system in the computation viewpoint.
The relationship between what is modelled in this viewpoint and what is modelled in the information viewpoint and enterprise viewpoint is not considered here (because the latter viewpoints are out of scope).

In essence the structure modelled in the computation viewpoint is whatever happens to be the appropriate combination of application components. As explained in clause 8, the term "application" has a very general meaning here, which includes all kinds of computer activity.

This formulation is specific to the SE-ODP and may not be general to ODP standardization with a wider scope (see clause 12).

15.2 Structure

No objects other than ODP components and ODP interfaces are visible in this viewpoint, and the latter are not necessarily always shown. By these criteria, no native components and no connexions to them are visible in this viewpoint.

The same information system may be modelled here at different levels of detail. An ODP component may be decomposed into further ODP components, which may in turn be decomposed into ODP components; and so on, until decomposition has reached the level of object granularity which the application designer wants to be visible to the SE-ODP. (Any further decomposition in the computations viewpoint would be a private matter, of no concern to the SE-ODP).

The object diagram in figure 10 shows, at a coarse level of detail, an example distributed application which happens to consist of human interfacing functions, application logic, and database access functions. Each of these groupings of function has been modelled as an ODP component (the corresponding ODP interface objects are not shown here).

![Diagram: human-computer interface (HCI) - application logic - database access]

**Figure 10 - An application example in the Computation Viewpoint**

Figure 11 shows this same example at a finer level of detail at which there are nine ODP components, of which six are application-specific and three are general purpose (the HCI, DBMS and filestore objects). This might be the final form model (or there might be further decomposition). If it were the final form model, the maximum degree to which this information system may be distributed via the SE-ODP is 9 separately executable software components, at up to 9 separate locations.
Figure 11 - A decomposition of the example in figure 10

As in all object diagrams, the relative position of the objects has no logical significance.

Using the recipe described in 13.6, any necessary native components are hidden by the ODP components (and are not visible in this viewpoint). E.g. the ODP component labelled "HCI ..." is probably realized via the screen and keyboard of a graphics workstation, but these are not visible here. Also, each ODP component in this viewpoint hides the language, operating system and hardware with which its realization is constructed.

15.3 Summary

As explained in clause 12, the SE-ODP has no need to "know" anything about matters such as human interfacing, database, file storage and application logic. In the example in figure 11, the SE-ODP would attach no connotation of "HCI" (nor any other connotation) to the object that is labelled "HCI". Likewise for the objects labelled "application logic", "DBMS" and "Filestore".

What the SE-ODP really needs to know about in this viewpoint is: which objects interact, via what ODP interfaces, and using which distribution transparencies.

16. DESCRIPTION IN THE ENGINEERING VIEWPOINT

16.1 General

This clause describes and explains what is visible in object diagrams that model an information system in the engineering viewpoint.

This viewpoint models the complete structure of using and providing the distribution transparencies required by the final form model for the computation viewpoint.

This formulation is specific to the SE-ODP and may not be general to ODP standardization with a wider scope (see clause 12).
16.2 Structure

The essence of the structure modelled in the engineering viewpoint is use of the transparent insertion recipe described in clause 13.5, to insert (arrangements of) processing components defined by distribution transparency recipes from the SE-ODP architecture (see clause 23).

NOTE 7:
The transformations described here are mainly mechanical, and in an implementation may be provided automatically by software tools.

The first transformation to be considered in the engineering viewpoint is concerned with extended distribution transparencies (defined in 23.2).

If a connexion between ODP components from the final form model of the computation viewpoint needs extended distribution transparencies, then extra ODP components to provide them (see 23.4) are inserted into the connexion. These inserted ODP components may be decomposed into more detailed ODP components. Any extended distribution transparencies needed by any of these inserted ODP components would require the insertion of further ODP components; and so on, until all needs for extended distribution transparencies are satisfied.

In the example shown in the object diagram in figure 12, one of the original ODP components might be the "application-specific DBMS access functions" in figure 11, and the other might be the "DBMS". The connexion between them might require concurrency transparency and failure transparency, and replication transparency, which might be provided by objects that support "atomicity" and object group abstractions (described in 23.4).

![Figure 12 - Provision of extended distribution transparencies](image)

Further transformations, described below, are concerned with providing the basic distribution transparencies (defined in 23.2).
For the final form model in the engineering viewpoint, each of these ODP components is decomposed into a pair of objects with a connexion between them. These paired objects are: an ODP component termed an interface adaptor, and a native component representing the rest of the original ODP component. The interface adaptor provides to the native component all the basic distribution transparencies required. For further description see 23.3.

Each interface adaptor is a part of the SE-ODP runtime support system (termed SE-ODP runtime), and has a connexion to the rest of SE-ODP runtime (the latter provides the means of interaction between interface adaptors). The structure of SE-ODP runtime is described in clause 24.

In the final form model of the engineering viewpoint, SE-ODP runtime would be represented at some appropriate level of granularity. Since the structure of SE-ODP runtime is general to all uses of the SE-ODP, it is often appropriate to represent the rest of SE-ODP runtime by a single object, the nucleus. The detailed composition and configuration of SE-ODP runtime could be shown separately in object diagrams general to all the information systems considered.

The object diagram in figure 13 is an example of the structure in the engineering viewpoint. It shows an information system consisting of pairs of objects that model fifteen separate processing components. But it gives no indication of object location relative to one another, nor any indication of which native components interact with which. Nine of the pairs of objects might represent the original ODP components from figure 11, and the other six might represent the inserted objects from figure 12. In this diagram it is not apparent which native components interact with which.

Figure 13 - Provision of basic distribution transparencies
As in all object diagrams, the size and relative position of the objects, and the orientation and length of the arcs between them, have no logical significance.
16.3 Summary

Further transformations would be needed to make explicit any differences of location. This would reveal some of the internal structure of SE-ODP runtime, as described in clauses 19, 20, 23 and 24.

The SE-ODP is largely about the extra inserted objects that are modelled in the engineering viewpoint. The original objects (from the computation viewpoint) are considered here primarily in terms of satisfying their distribution transparency needs.

17. DESCRIPTION IN THE TECHNOLOGY VIEWPOINT

17.1 Introduction

This clause describes and explains what is visible in object diagrams that model an information system in the technology viewpoint.

This viewpoint models use of native components to support the final form model from the engineering projection, and is not considered in detail by SE-ODP standardization.

This formulation is specific to the SE-ODP and may not be general to ODP standardization with a wider scope (see clause 12).

17.2 Structure

Only those ODP components from the engineering viewpoint that decompose to reveal native components of interest need be considered. The general approach for revealing the native component(s) from which an ODP component is constructed has been described in clause 13.6 and shown in figure 9.

The question of "what native components are of interest?" needs further study, but some preliminary guidelines are given below. The main candidates for decomposition in this viewpoint are:

(a) the SE-ODP interpreter, which hides processors and memory (see clause 19);
(b) an SE-ODP stable-storage object, yet to be defined, which would hide non-volatile storage, such as discs; and
(c) the SE-ODP IPC object which hides communications mechanisms (see clause 20).

This corresponds to the general threefold classification of distributed processing resources into processing, storage and communication.

There is also a need to distinguish native components that are basic to separation in space and time. The candidates are a native component for modelling units of physical separation ("physical units"), and a native component for modelling clocks. For practical reasons a "physical unit" would generally be in one-to-one correspondence with a local clock (i.e. the accessibility of a local source of timestamps sets the bounds of a physical locality).
Encryption mechanisms may be another kind of *native component* which is of fundamental significance.

There are various *processing components* such as the terminal devices, sensors and actuators via which an *IT system* interacts with the people, *natural systems* and *technical systems* in its environment. As explained in clause 12, standardization of such mechanisms, and detailed concern about them, is outside the scope of *SE-ODP* standardization. A *native component* sufficiently general to model all of these mechanisms would be a "*transducer object*", operating across the boundary between the *IT system* and the physical world outside it.

Another boundary occurs between the conceptual world of *ODP components* and the actual executable specifications (i.e. programs, written in computer languages). A *native component* sufficiently general to model all of the latter would be a "*denotation object*".

17.3 Summary
In the *technology viewpoint*, it may be sufficient for *SE-ODP* standardization only to consider *native objects* pertaining to *SE-ODP runtime* (and not those pertaining to the *application components*). If so, this would be a considerable simplification. Further study is needed.

18. SE-ODP OBJECT INTERACTIONS

18.1 Introduction
This clause describes the structure of *interactions* via *ODP interfaces*.

18.2 Overview
As explained in clause 13, most *processing components* considered in *SE-ODP* standardization are modelled as *ODP components* between which there are *connexion* via *ODP interfaces*.

A common *model of interaction* semantics is applicable to all *ODP interfaces* (see 18.3), and thereby to all *ODP components*. Any language used to define *ODP interfaces* is required to conform to this Semantic Model (see 18.4). These "interface definition languages" would be used to define operation signatures, partial ordering constraints, *distribution transparency* constraints, etc.

This unification of *interaction* semantics is summarized in figure 14.

*NOTE 8:*
Figures 14 and 15 are examples of how the object diagram notation may be used to model abstract joint action between specifications.
Figure 14 - Harmonization of Interface Definition Languages (IDLs)

Any of these interface definition languages may be used to specify an ODP interface. The resultant interface description is defined without reference to representations of the interface (see 18.5). Each ODP interface is therefore an abstract interface conforming to the common Semantic Model, and has many possible program representations (see 18.6) and many possible transfer syntax representations (see 18.7). This flexibility is summarised in Figure 15.

Figure 15 - Diversity of Interface Representations

The freedom to independently vary programming interfaces and transfer syntax is supported by the interface adaptor architecture which is described in 23.3.3 and illustrated in figure 19. There is almost complete decoupling from technology-dependent detail (see 18.8).

These various characteristics are now each considered in more detail.
18.3 Interaction Semantics

In order to reason unambiguously about complex distributed processing systems, it is first necessary to define a Semantic Model of distributed computation, specifically the semantics of computational interactions.

Existing models of computation (e.g. the von Neumann model) were developed to describe computing in non-concurrent, single-processor environments. Specification languages and programming languages based exclusively on such models have inadequate power for expressing ideas of concurrency and distribution. In addition, all except the developing object-oriented languages have difficulty in representing encapsulation adequately. For SE-ODP purposes, there should be one model of interaction semantics. This would embody all the concepts necessary to completely describe interactions between the components of distributed processing systems, including notions of encapsulation, concurrency, distribution transparency and atomicity.

Clause 29 outlines a standardization work item on this subject.

18.4 Languages for Definition of Object Interactions

To provide notation in which to specify interactions, some language representation of the Semantic Model (see 18.3) is necessary. There may be several such "interface definition languages", but each must conform to the same Semantic Model if there is to be consistent interpretation of their semantics.

The purpose of these "interface definition languages" is specification of interactions between objects. This is different from the traditional role of programming languages.

Many different kinds of language-design choices can be made when designing a language (or notation) for this purpose. Each choice has different trade-offs (e.g. maximizing opportunities for static checking by language systems, or maximizing re-use of existing notation).

The Interface Definition Notation (IDN) in the ECMA-127 RPC standard is an example of an "interface definition language" at this level.

18.5 Abstract Interface

An interaction between ODP components is constrained by an abstract interface, which is specified in some "interface definition language" (see 18.4) that conforms to the Semantic Model (see 18.3). This provides a complete definition of the interaction, as visible in the computation viewpoint.

This specification is abstract, in that it is uncommitted to any particular choice of concrete form. Such choices are made separately in the engineering viewpoint (see 18.6 and 18.7) and in the technology viewpoint (see 18.8).

18.6 Application Programming Interfaces

The application programmer's view of an interaction is via some programming interface which is made visible in the engineering viewpoint.

Such an interface has a concrete form that is necessarily committed to some particular syntax, control structure and language binding (e.g. to the syntax and
semantics of some particular programming language, and perhaps to using a control structure of procedure calls for external interactions). These concrete interfaces are visible as the interfaces of *native components* (see 23.3.3). Each is an *engineering viewpoint* representation of the corresponding *abstract interface* that was defined in the *computation viewpoint*.

For the purpose of interworking, the programming interface is essentially a local matter, in that it is not directly visible to the *ODP component* at the other end of the *interaction*. Therefore, in principle, there may be arbitrarily many different programming interfaces for a given *interaction*, provided that each is a representation of the same *abstract interface*.

For the purpose of software portability, there is a case for using, wherever possible, the same programming interface for a given *interaction*.

This de-coupling of the choice of programming interface from other characteristics of the *interaction* is an important contribution to simplifying system design and development.

18.7 Transfer Syntax

The transfer syntax is the way in which the structure and content of the *interaction* are represented and encoded in transit between the *application components*. This is made visible in the *engineering viewpoint*.

The transfer syntax may be different from the syntax of the programming interfaces. Also any necessary distinctions between abstract transfer syntax and concrete transfer syntax are made at this juncture.

Arbitrarily different transfer syntaxes may be applicable in different circumstances (e.g. when *objects* are co-located, when *objects* are in separate computers of the same kind, when the computers are different, and when different kinds of interconnection are used). Both ends need to make mutually acceptable choices of transfer syntax and to do any necessary conversions.

The specifications of the transfer syntax and the programming interface provide a complete definition of the representations of the *interaction* which are visible in the *engineering viewpoint*. The semantics and overall syntactic structure continue to be as defined in the *computation viewpoint*.

18.8 Technology-dependent Structure

The *interaction* is realized by means that are modelled in the *technology viewpoint*.

Wide freedom of choice at this level is allowed by the completeness of the abstract definition of *interaction* behaviour in the *computation viewpoint* and *engineering viewpoint*; see 18.3 to 18.7.

Furthermore, the actual structure at this level is necessarily implementation-dependent; e.g. different *operating systems* have different process, memory and communications structures; different computers represent stored values in different ways; different kinds of interconnection have different architectures and
protocol structures; different infrastructure implementations have different interfaces, different concurrency provisions.

Therefore, the interaction visible in the technology viewpoint is not defined here; its heterogeneity is not constrained by SE-ODP architecture. This ties in with the role of native components in the technology viewpoint, introduced in clause 17.

18.9 SE-ODP Prescriptions

For SE-ODP purposes, there are the following prescriptive constraints on the specifications of interactions via ODP interfaces.

(a) One standard Semantic Model shall be used throughout.
(b) Standard interface definition language(s) shall be used. For each there is necessarily an agreed mapping onto the Semantic Model (a).
(c) For particular SE-ODP functions, standard abstract interfaces shall be used (e.g. for interaction with the SE-ODP trading function).
(d) Particular standard programming interfaces may be required in some cases.
(e) Particular standard transfer syntaxes (e.g. ISO 8824 and 8825) and protocols may be required where interactions are remote and use OSI.

These matters are considered in more detail in 28.4, and in the clauses in Section Five.

19. SE-ODP PROCESSING MODEL

19.1 Introduction

This clause provides a preliminary description of the SE-ODP processing model which relates together processing, memory and communications. This is an engineering viewpoint matter.

19.2 Definitions

The following definitions apply.

processing: programmed activity executed in computers.

SE-ODP processing model: a model of how processing is structured and organized in the SE-ODP.

interpreter: a mechanism that realizes processing.

SE-ODP interpreter: an interpreter that realizes SE-ODP processing model concepts.

native interpreter: an interpreter that is a native component.

capsule: an object that is specific to the SE-ODP, and models the computer address space within which processing is confined by interpreters.
19.3 General

Different computer languages, operating systems and computer hardware have different processing models. All have much in common, and use much the same terminology (although often with undeclared differences of meaning).

To achieve consistent understanding amid this heterogeneity, it is necessary to define a common set of processing concepts. This is what the SE-ODP processing model does.

To achieve distributed processing consistently in heterogeneous processing environments, it is necessary to define some common mechanism to support these SE-ODP processing model concepts. This is what the SE-ODP interpreter does.

Processing is confined within capsules, and all SE-ODP visible interactions between capsules is via SE-ODP IPC (see clause 20).

19.4 Processing model

Processing is organized around the three functional areas of processing, memory and communication. The encapsulation, concurrency and synchronization characteristics of processing are of particular importance to distributed processing. The SE-ODP processing model may also include particular concepts to support interaction constraints (e.g. distribution transparency and atomicity, which would be defined in the Semantic Model).

Minimality, simplicity and practicability should be guiding principles for design of the SE-ODP processing model. The details are for further study.

19.5 SE-ODP Interpreter

The SE-ODP, and the software of the applications using it, are ultimately mechanized by processing provided directly by the local environment (i.e. the instruction set of some native interpreter).

However, some of the SE-ODP functions offered to application components derive from the SE-ODP processing model. Figure 16 is an object diagram which shows the role of the SE-ODP interpreter (in the engineering viewpoint) and the relationships to the native interpreter (in the technology viewpoint).
visible in the
Engineering Viewpoint

native components
of the application,
which are hidden by
ODP components

SE-ODP runtime,
including all the
interface adaptors

visible in the
Technology Viewpoint

all executable
denotation

native
translator

Figure 16 - The SE-ODP Interpreter

The SE-ODP interpreter provides specialized primitives for concurrency, synchronization, inter-process communication, etc. In some cases these primitives can be mapped directly onto normal local processing. But in other cases the SE-ODP interpreter provides the necessary function itself. This would usually be done by using some combination of functions provided by the native interpreter (including the local operating system) and other parts of the SE-ODP. The details are for further study.

20. SE-ODP COMMUNICATIONS MODEL

20.1 Introduction

This clause describes the Communications Model used for all interactions between processing components that are executed in separate computer address spaces (capsules).

20.2 Definitions

The following definitions apply.

inter-process communication (IPC): communication between processing in separate computer address spaces (in the SE-ODP IPC case these would be capsules).

SE-ODP IPC: IPC relating to the SE-ODP processing model.

20.3 Description

The Communications Model exists only to supports the SE-ODP processing model and the SE-ODP interpreter. They formulate all the relevant interactions in terms of inter-process communication, which the SE-ODP interpreter supports with IPC primitives and synchronization and scheduling primitives. Therefore, this Communications Model is exclusively concerned with supporting SE-ODP IPC.
The IPC primitives are required to provide a normalized basis for mechanization of all the external interactions (including those needing isochronous and multicast communication). This normalization, which masks the diversity of actual communications mechanisms, is visible in the engineering viewpoint. The diversity of underlying mechanisms is to varying degrees visible in the technology viewpoint.

The means of communication is presented to the rest of the SE-ODP as services provided by an SE-ODP IPC object, which hides the diversity of the native components which it uses to achieve the required communication. These diverse objects provide diverse communications services for local IPC, data telecommunications, voice telephony, etc. This structure is illustrated in figure 17.

![Figure 17 - SE-ODP communications](image)

The real-world diversity of communications technology includes many kinds of interconnection that do not conform to open standards. It is necessary that SE-ODP architecture has descriptive flexibility for coverage of this heterogeneity (as illustrated by the example in figure 17). This does not preclude prescriptive formulation of ODP standards which may require that appropriate OSI interconnection standards are used.

21. APPLICATION CONFIGURATION

21.1 Introduction

This clause describes concepts for the configuration of distributed information systems, and applies this to some of the examples given in clauses 15, 16, 17. Configuration is to be the subject of more detailed ECMA standardization; see clause 28.

21.2 Definitions

The following definitions apply.

*NOTE 9:*

These definitions are derived directly from the object model. They depend on the particular definitions of "activity", "object", "interaction" and "action" provided in clause B.6.
Basic service concepts derived from the object model:

**service**: activity for co-operative use by other objects.

**service interaction**: an interaction in which one object provides a service for use by the other object.

**exporter**: an object to which provision of a service is attributed.

**importer**: an object to which use of a service is attributed.

**export**: a proposal to provide some particular service.

**import**: a proposal to use some particular service.

Configuration concepts derived from the basic service concepts:

**client-server interaction**: a service interaction in which the importer initiates action.

**producer-consumer interaction**: a service interaction in which the exporter initiates action.

**client**: an object that is importer with respect to the client-server interaction considered.

**server**: an object that is exporter with respect to the client-server interaction considered.

**consumer**: an object that is importer with respect to the producer-consumer interaction considered.

**producer**: an object that is exporter with respect to the producer-consumer interaction considered.

**client-server model**: a model of object configurations in which the connexions considered are identified with client-server interactions. May be abbreviated to client-server.

**producer-consumer model**: a model of object configurations in which the connexions considered are identified with producer-consumer interactions. May be abbreviated to producer-consumer.

Configuration support concepts:

**trading**: activity pertaining to imports and exports.

**trading service**: a service used to match imports and exports.

**trader**: an object to which provision of a trading service is attributed.

### 21.3 Using and Providing Services

 Clause 15 has described how information systems are modelled in the computation viewpoint as ODP components with connexions between them.

The main relationship between these ODP components is that they cooperate with one another via the provision and use of services. This is modelled by the connexions between them. Each connexion is identified with some service interaction for which one of the ODP components is the exporter and the other is
the *importer*. The contribution that each *ODP component* makes to the *information system* is defined by what it *exports*.

These same kinds of relationships also prevail in the *engineering viewpoint* and the *technology viewpoint*, where (as explained in clauses 16 and 17) different *objects* are visible.

**21.4 Client-Server and Producer-Consumer Models**

The *client-server model* and the *producer-consumer model* are particular formulations of the general *importer / exporter* structure. Arrangements of *ODP components* can be modelled in terms of *client, server, producer* and *consumer* roles.

The terms *importer, exporter, client, server, producer* and *consumer* are not strictly applicable to the *object* as a whole. An *object* may have several of these various roles with respect to its several *connexion*. But it is often convenient to refer informally to the *object* itself as the "*client*" or "*server*" in the context of some particular *connexion*.

Some of the long standing disputes over "what is a server?" can now be resolved by making distinctions between *viewpoints*. In the *computation viewpoint*, the term *server* refers to application structure (and applications experts have corresponding notions of what a *server* is). In the *engineering viewpoint*, *server* refers to *SE-ODP* support structure; and in the *technology viewpoint*, *server* refers to the artifacts from which the *information system* is constructed (so various kinds of *operating system* experts have various other notions of what a *server* is). Similarly, any usage of these concepts in the *enterprise viewpoint* or *information viewpoint* is likely to be different. These same concepts can also be applied at many different levels of granularity (i.e. a *server* is not necessarily only something that is visible as a unit of *distributed system* granularity).

**21.5 Trading**

These configurations of *objects* providing and using *services* are organized via what is termed *trading*. This operates at the level of individual *objects* and *connexion*, organized within larger scale structure which integrates with concepts of "management domains" etc.

The *SE-ODP* provides a *trading service* via *trader objects*, which are themselves configured as a Trading Application. Clause 24.4 explains how this *distributed application* is positioned as the *trading utility*. Appendix D provides a preliminary description of *trading*.

**21.6 Generics**

Standard *ODP interfaces*, standard *ODP components* and standard configurations of them could be defined for commonly occurring functions (e.g. for file services, document storage and retrieval, printing, electronic mail services, configuration management, etc.). Some of these items would be general to all *distributed processing*, some would be domain specific, and many are already the subject of open standards. In the latter case, further standardization work would
be needed to adapt existing functions to take advantage of ODP architecture and technology.

These matters are actually outside the scope of SE-ODP standardization; but may varyously be within the scope of ODP standardization more generally (see clause 12).

21.7 Review

These concepts of providing and using services, and consequent client-server relationships, are applicable to the descriptions already given in clauses 15, 16 and 17. This further level of structural detail can now be added by refinement of the object diagrams in those clauses.

In figure 10, the "HCI" object is a client of the service that is exported by the "application logic" object, which in turn is a client of the service that is exported by the "database access" object. In figure 11 these objects are decomposed into finer grained objects between which there is a further level of trading; e.g. there is an "application logic" object that exports a service labelled E to three other "application logic" objects. These examples are modelled here in the computation viewpoint, and each object considered is an ODP component.

In figure 12, extra ODP components are inserted into this configuration, in order to provide extended distribution transparencies. ODP component₁ is the importer of a service which is apparently provided by ODP component₂; but this service is augmented transparently by the inserted ODP components. In effect, the latter collectively masquerade as the server to ODP component₁ and as the client to ODP component₂. Among themselves they have various client-server relationships. (This example also indicates a need for care about security characteristics when the SE-ODP inserts objects transparently to achieve desired characteristics, as modelled in the engineering viewpoint.)

Various client-server relationships could be attributed to the more detailed engineering viewpoint structure illustrated in figure 13. This requires further study.

The above example did not include any producer-consumer interactions. These are relatively common in applications such as process-control and Communications Command and Control.

22. SE-ODP NAMING & BINDING

22.1 Introduction

This clause provides a preliminary description of SE-ODP naming and binding requirements (not the solutions, because at this stage the subject has not yet been studied in detail).

22.2 Definitions

The following definitions apply.

name: a symbol by which objects may be identified.

ODP name: a name that is constrained by SE-ODP standardization (see note).
native name: a name that is not constrained by SE-ODP standardization (see note).

NOTE 10:
The definitions of "ODP name" and "native name" refer to "SE-ODP standardization" not "ODP standardization". The latter might include standardization of names considered here to be "native names". When ODP standardization is at a more advanced stage of development, these particular definitions might need to be reformulated.

naming context: a context within which a name has an agreed interpretation.

context relative name: a name which is interpreted within an identified naming context.

naming tree: a hierarchy of context relative names, each having an unique value within its immediate naming context.

binding: a mapping between a name and an object.

address: a name for which there is a binding to an object that represents location.

early binding; static binding: binding in which the mapping is determined in some previous epoch, and persists thereafter.

late binding; dynamic binding: binding in which the mapping is determined only in the epoch considered (which is usually the "execution epoch").

ODP association: a binding between an ODP component and a name used by another ODP component.

22.3 Naming

In ODP there is an immense diversity of names. Different considerations apply to names in the different viewpoints, and in the many different technical disciplines that are involved in ODP. Many of the names that are encountered are native names arising in heterogeneous social systems and technical systems that are not constrained by ODP standardization.

Therefore, the SE-ODP approach to naming has to respect this diversity and avoid being unduly prescriptive.

All names are inherently context relative names (i.e. any name necessarily has to be interpreted within some agreed naming context). In some cases this naming context may be considered to be "global". But many other different naming contexts may also be considered to be "global" (e.g. in different enterprises, in different technical disciplines, and in separately constructed systems and subsystems). The SE-ODP therefore has to operate in a world where naming trees do not all have a common root. Links will be continually forming between existing naming trees, but new separate naming trees will continually be forming elsewhere. This reflects the continual change that is characteristic of social systems and complex technical systems. Many naming trees will deliberately be kept separate for security reasons, or because there is no reason why they should be related (e.g. the names of organizational roles in some enterprise, and the names of software components elsewhere, are inherently unrelated).
Likewise, the SE-ODP has to be tolerant of diverse ways of representing the values of names.

However, the SE-ODP should also be able to take advantage of areas in which naming is "global" and the name representations are homogeneous (e.g. names conforming to ISO registration schemes).

A further complication is that the same name may bound to many objects; and vice versa.

Addresses are similarly diverse (e.g. "network addresses" and "computer memory addresses"). It is likely that addresses will be treated as native names, originating outside the scope of SE-ODP standardization.

Therefore, the characteristics likely to be defined for ODP names are: context relative names, multiple independent naming trees, with ways of forming links between naming trees (probably by introducing common roots, or cross-links, or aliases), and ways of achieving optimization where names are homogeneous and "global".

22.4 Binding

As with naming, many different kinds of binding are relevant to the SE-ODP, and much is beyond the reach of SE-ODP standardization (e.g. the bindings internal to the program structure of a native object).

The SE-ODP makes some important distinctions between the use of static bindings and dynamic bindings (see 23.3.4).

ODP associations are a particularly important kind of binding. They are usually formed via trading in some epoch (see clause 21 and Appendix D).

An area requiring further study is the relationship between connexions, and ODP associations and communications connections. Figure 18 illustrates a connexion for which these relationships are now considered.

![Figure 18 - An object diagram](attachment:image.png)

The presence of the connexion in the object diagram does not model the presence of an ODP association between the objects. At most it expresses constraints by which it is possible that one or more ODP associations exist. If one does exist, it may be a dynamic binding that only exists for some of the time.

A further point is that the corresponding ODP association(s) are not strictly between the ODP components. The ODP association would be between ODP component₁ and some name known to ODP component₂, such that ODP
component\textsubscript{2} can reference ODP component\textsubscript{1}; or vice versa; or both these bindings might exist.

The presence of a connexion in the object diagram and the existence of an ODP association do not model or require the existence of a "connection" for communication between the ODP components, nor the existence of any other kind of communications channel. At most they indicate that a communications channel should exist and be available for this purpose when interaction between the ODP components occurs. If the ODP components are in separate address spaces (defined as capsules in clause 19), then this becomes a matter for inter-process communication via SE-ODP IPC (see clause 20), but not otherwise.

Some of these distinctions are made by modelling in different viewpoints. The connexion in figure 18 is probably visible in the computation viewpoint (but such ODP components might be in the engineering viewpoint, e.g. as in figure 12). Any trader and SE-ODP IPC involvement would be modelled in the engineering viewpoint, and any underlying communications connection would probably be a matter for the technology viewpoint. But the ODP associations may be considered to be detailed program implementation matters, not visible in object diagrams representing system structure.

A further point is that the presence of an ODP association does not necessarily imply a binding between state in the separate ODP components. This raises wider issues (e.g. "atomicity") that are not considered further here. Certain provisions for bindings to the state of remote components are defined in ECMA-127, and these are likely to be included in future SE-ODP provisions.

The preliminary conclusion is that great care is needed in relating binding concepts to particular mechanisms such as ISO 8649 ACSE associations, ISO 9072/1 ROSE BIND primitives, and ECMA-127 RPC module linkage.

23. SE-ODP DISTRIBUTION TRANSPARENCY TECHNIQUES

23.1 Introduction

Distribution transparencies have been defined and explained in the ISO RM-ODP work that is summarised in B.4.

The insertion of distribution transparency recipes into models in the engineering viewpoint has been described in clause 16. The distribution transparencies inserted are those required for interactions between (the realizations of) ODP components (which usually model application components).

This clause describes the distribution transparency recipes that are inserted.

\textit{NOTE 11:}

Clause B.4 of Appendix B should be consulted at this point because it provides the base definitions and base concepts of distribution transparency.
23.2 Definitions

The following definitions apply.

**basic distribution transparencies**: access transparency and location transparency (see note).

**extended distribution transparencies**: concurrency transparency, replication transparency, failure transparency and migration transparency (see note).

*NOTE 12:*
The above definitions should be updated when the definitive set of distribution transparencies has been agreed in the RM-ODP (see B.4).

**interface adaptor**: an object that provides basic distribution transparencies to an ODP component.

**ACID properties**: a set of characteristics prescribed in transaction processing (see ISO 10026 and 23.4.2).

**atomic object**: an object to which ACID properties are attributed.

**object group**: a collection of inter-related objects, for which there is an abstraction that is a single object.

**object factory**: an object that creates instances of objects from templates that specify the required objects.

23.3 Basic Distribution Transparency Techniques

23.3.1 General

Together, the basic distribution transparencies support a style of interaction that is independent of whether ODP components are co-located or remote.

The basic distribution transparencies also provide independence from the heterogeneity of the native objects by which ODP components are realized, and provide independence from the heterogeneity of underlying networks, etc.

To simplify description, these basic distribution transparencies are explained here without reference to them being selective, which is considered in 23.5.

23.3.2 Unified Interaction Semantics

The foundation for distribution transparency is that the same Semantic Model is applicable to all interactions between ODP components, irrespective of whether the objects are co-located or remote. This unification of interaction semantics is described in clause 18.

23.3.3 Interface Adaptors

The realization of each ODP component that may use distribution transparencies is modelled as a native component and an interface adaptor, both of which are visible in the engineering viewpoint (see 16.2).

The interface adaptor is part of the SE-ODP runtime support structure (termed SE-ODP runtime) which is described in clause 24. It mechanizes any necessary dynamic binding, data conversion, use of buffers, and marshalling of data between buffers and program data structures, etc.
NOTE 13:
The term "stub" is widely used to refer to this subject area (e.g. the "RPC stubs" described in ECMA 127). But "stub" has implementation-specific connotations, and the term is not used in this Technical Report. The term "interface adaptor" is used instead, with a more general definition.

The role of an interface adaptor is illustrated in figure 19. The interface adaptor ensures that the native component sees only its local interaction (A) with the interface adaptor, and does not see the internal mechanisms of the interface adaptor, nor the interactions (B) between the interface adaptor and other objects. Similarly, no other objects see the interaction (A) or the internal mechanisms of the interface adaptor. See clause 13.6 for description of the general principles of this information hiding.

![Figure 19 - General role of an interface adaptor](image)

The original ODP component (e.g. application component) may have connections to multiple ODP components. In which case the interaction A (and perhaps B) would be a composition of the individual interactions, and the interface adaptor would likewise be composite. In more detailed object diagrams this structure could be decomposed into its elements; i.e. separate elementary interactions $A_1 \ldots A_n$ via separate elementary interface adaptors.

As explained in clause 18, the details of the local interaction A may be implementation-specific (i.e. particular to the native component), although there is necessarily an equivalence to the abstract interaction defined by the ODP interface concerned. The interface adaptor hides this local variability, i.e. heterogeneity.

An interface adaptor is, in effect, that part of SE-ODP runtime which is customized to the needs of a particular realization of an ODP component. It supports whatever local interaction (A) the native component uses to represent interactions concerning the ODP component.

The realization of the interface adaptor and the native component are executed in the same address space (defined as a capsule in clause 19). The interface adaptor acts as a local "proxy" that represents ODP components which are external to the native component and with which the native component has interactions. The native component interacts with the interface adaptor as if it (the interface adaptor) were the external object. In this way, external interactions can be programmed as local inter-module interactions within the local address space. Therefore, the local interactions (A) are
usually those of a normal inter-module programming interface, not a communications programming interface.

23.3.4 Transparent Dynamic Binding

The bindings between ODP components may be static bindings or dynamic bindings. The native components revealed by decomposition of the ODP components participate in these same bindings. For many kinds of information systems it is vitally important (e.g. for reasons of design stability, integrity and security) that bindings at this user-visible level are static bindings that are guaranteed not to change.

However, the bindings between interface adaptors are always dynamic bindings; i.e. they are actually made and unmade at runtime (even if always between exactly the same instances of the same interface adaptors). This binding structure is illustrated in figure 20.

![Diagram](image)

**Figure 20 - Transparent dynamic binding**

This underlying dynamic binding is a source of flexibility and adaptability which is fundamental to the provision of distribution transparencies (e.g. it provides opportunities for location transparency). This dynamic binding is an internal mechanism that is hidden from the native components by the basic distribution transparencies; i.e. it need not be visible in the local interaction (labelled A in figure 19) between a native component and its interface adaptor.

All the SE-ODP binding functions use the trading service to match proposals to provide and use services. Trading is described in Appendix D.

23.4 Extended Distribution Transparency Techniques

23.4.1 General

The extended distribution transparencies exploit opportunities for qualitative improvements that arise when basic distribution transparencies are provided.

ECMA has not yet studied this area of ODP in detail. Therefore, this is only an outline description of subjects for further study.

To simplify description, these extended distribution transparencies are explained here without reference to them being selective, which is considered in 23.5.
23.4.2 Atomic Objects

*Interactions* between *ODP components* may be required to be robust to failures.

The *basic distribution transparencies* provide a degree of robustness to communications failures, but not coverage for other failures (e.g. failure of the *ODP components* themselves).

Support for *ACID properties* is a more general solution, which is relevant to all the *extended distribution transparencies*. *ACID properties* are concerned with the atomicity, consistency, independence and durability of the effects of *actions*. ISO 10026 (OSI TP) is the subject of current ECMA and ISO work in this area.

A prototype architecture for automated *ODP* support of *atomic objects* via *extended distribution transparencies* is described in [ANS A 89].

23.4.3 Object Groups

*N O T E  1 4:*  
This description considers "execution groups" composed from *ODP components* that offer the same *ODP* interface(s) and identical behaviour. Other kinds of object groups are for further study (e.g. grouping for management purposes).

*Object groups* are a basis for functional distribution, replication and parallelism which provide opportunities for improved performance, availability and fault tolerance.

The improved performance may be gained via parallel execution using separate resources and by load balancing across these resources. Improved availability and fault tolerance may be gained via multiple *objects* that provide alternative resources, or hot standby, or majority voting of results from suitably independent sources.

This multiplicity of replicated *objects* and the coordination of them (although intended to be ultimately beneficial) adds a major degree of complexity.

The main sources of this added complexity are: the sheer numbers of *objects* involved; the continually changing membership of each collection of *objects* (due to failures, reconfiguration of resources, etc.); the difficulties of organizing *joint action* within each collection of *objects*; the difficulties of achieving coherent *joint action* between the separate collections of *objects*; compounded by the need for optimizations to ensure that the *interactions* can achieve performance goals, and without violating integrity constraints; the scaling-up of all of this for many *interactions* between many collections of many *objects*; plus provision of means of (re-)configuring and managing the resultant complex.

Comprehensive and automated support for *object groups* is a way of selectively hiding this complexity (the multiple *objects* in each *object group* behave towards other *objects* as if the multiple *objects* were collectively one *object*).
A prototype architecture for automated ODP support of object groups via extended distribution transparencies is described in [ANSA 89]. There is a close relationship between the latter and the prototype architecture to support real-time fault-tolerant ODP which is described in [DELTA-4, 88].

For some fields of application these object group concepts and related fault-tolerance may seem rather esoteric (although for others they are a vital necessity). A universally applicable case in which they are vitally necessary is that of the SE-ODP utilities, which are an integral part of SE-ODP runtime (see clause 24). Therefore, support for object groups is necessarily a fundamental characteristic of the SE-ODP.

23.4.4 Object Factories

The objects that are the realizations of ODP components at runtime have to be created and destroyed, and sometimes re-located.

The object factory concept includes the creation of objects, and perhaps their re-creation when they are re-located. It may also have a role in their ultimate destruction.

There are related concepts of the "freezing" and "thawing" of objects that become inactive and then become active again at some later time. A "frozen" object would be modelled as having a connexion to some kind of stable-storage object and not to an interpreter. A "thawed" object would be modelled as having a connexion to an interpreter that can execute it.

These mechanisms for the creation and destruction of objects (and "freezing" and "thawing") may have a role in supporting relocation transparency.

A prototype architecture for automated ODP support of object factory functions within provisions for extended distribution transparencies is described in [ANSA 89].

23.5 Selective Distribution Transparencies

At the level of object granularity considered in the SE-ODP, the basic distribution transparencies are always available, but may be selectively used. For a native component to be selective about its use of these distribution transparencies, the local interaction (labelled A in figure 19) between the native component and the interface adaptor would need to include means to determine location, or to participate in trading, etc.

As explained in clause 15, objects that provide the extended distribution transparencies are only inserted where needed. Therefore the extended distribution transparencies are inherently selective. Furthermore, the ODP interfaces of the ODP components inserted may be specified to reveal distribution characteristics to the native objects concerned, and to provide means of manipulating these characteristics.

Therefore, all use of distribution transparencies is selective, according to the requirements of the distributed application concerned.
23.6 Distribution Transparency when Co-located

ODP components may be co-located in the same computer. As in all other cases, such ODP components are modelled (in the engineering viewpoint) by a native component and its interface adaptor.

Depending on the degree of separation required, native components co-located in the same computer ("physical unit") are either positioned in separate address spaces (defined as capsules in clause 19), or in the the same capsule. For each native component its interface adaptor is also in the same capsule.

(In an implementation, any interactions directly between the native components in the same capsule may be realized via direct inter-module linkage. But this is an implementation optimization that is not visible in models in the engineering viewpoint, and probably not in the technology viewpoint either.)

Co-located ODP components are not necessarily homogeneous; i.e. they may be designed and implemented separately, by different organizations, at different times, and using different programming languages, house styles and module interfaces (i.e. different "A"s in figure 19). In this respect co-located objects are no different from objects that are physically separated. In all cases the basic distribution transparencies (and in particular access transparency) hide the differences arising from heterogeneity.

24. SE-ODP RUNTIME SUPPORT

24.1 Introduction

This clause describes the SE-ODP in terms of run-time support structure. This structure is visible in the engineering viewpoint and the technology viewpoint.

24.2 Definitions

The following definitions apply.

SE-ODP runtime: a distributed IT system which provides ODP distribution transparencies, to distributed applications, at run time.

nucleus: a part of SE-ODP runtime which is the means of interaction between interface adaptors.

DT0; DT1; DT2; DT3: classifications of objects with respect to their provision and use of distribution transparencies (see Table 1).

SE-ODP utility: any part of SE-ODP runtime which is modelled as one or more DT3 objects.

distribution transparency utilities: SE-ODP utilities which provide distribution transparencies.

trading utility: an SE-ODP utility for matching proposals to provide and use services.

security utilities: SE-ODP utilities to provide security facilities.

management utilities: SE-ODP utilities to provide management of SE-ODP runtime.
object factory utilities: SE-ODP utilities to provide object factory functions.

SE-ODP logical unit: an object which represents location to SE-ODP runtime.

24.3 Classification w.r.t. Distribution Transparencies

For description of SE-ODP runtime, all objects are classified with respect to (w.r.t.) distribution transparencies, per table 1 (which uses truth table notation; i.e. 0 = No, 1 = Yes). For example, an object is classified as D2 if it is a user of distribution transparencies, but is not a provider of distribution transparencies.

<table>
<thead>
<tr>
<th>role</th>
<th>DT3</th>
<th>DT2</th>
<th>DT1</th>
<th>DT0</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>provider</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1 - Classification w.r.t. distribution transparencies

The role of SE-ODP runtime is to provide distribution transparencies. Therefore, all objects that are part of SE-ODP runtime are considered to contribute to the provision of distribution transparencies, and are necessarily classified as DT1 or DT3.

Similarly, the ODP components modelled in the computation viewpoint are necessarily classified as DT2. The classification DT0 would be applicable to objects modelled in the information viewpoint (where distribution transparencies are not known about).

Clause 15 has already described the case in which objects to provide extended distribution transparencies are modelled in the engineering viewpoint as users of distribution transparencies (just like objects modelled in the computation viewpoint). The former are DT3 objects, and the latter are DT2 objects.

In general, a DT3 object cannot make recursive use of the distribution transparency that it itself provides. SE-ODP runtime can therefore be modelled as a hierarchy of objects with DT1 objects providing the basic distribution transparencies. The DT3 objects that use these distribution transparencies can provide other supportive functions and richer distribution transparencies. Other DT3 objects can use these; and so on until the full range of SE-ODP runtime functions (e.g. extended distribution transparencies) is provided.

To facilitate distributed implementation of SE-ODP runtime, much of it is defined as SE-ODP utilities i.e. DT3 objects (see 24.4).

24.4 Functional Decomposition

All the structure considered in this subclause is viewed in the engineering viewpoint.
The structure of SE-ODP runtime viewed with respect to some particular interaction is modelled by the object diagram in figure 21. The nucleus provides support directly to the interface adaptors (see 23.3.3).

Figure 21 - An interaction via SE-ODP runtime

The nucleus uses SE-ODP IPC (see clause 20) for all inter process communication, and the SE-ODP interpreter (see clause 19) for execution.

Most of the rest of SE-ODP runtime consists of SE-ODP utilities, as follows.

The trading function is modelled as the SE-ODP trading utility. This utility exploits the basic distribution transparencies so that trading functions can readily be distributed. It exploits the extended basic distribution transparencies so that trading functions can be replicated and located close to where they are used, and can have high degrees of parallelism and fault tolerance.

Likewise, there are distribution transparency utilities, security utilities, management utilities and object factory utilities; all of which may be implemented as highly replicated distributed applications.

The general functional decomposition of SE-ODP runtime is modelled by the object diagram in figure 22.

Figure 22 - General functional decomposition of SE-ODP runtime
This positioning of SE-ODP runtime functions as SE-ODP utilities also achieves complete modular separation of each from all of the others, and allows the nucleus to be correspondingly smaller and simpler. The hiding of all communications by SE-ODP IPC also removes from the nucleus another major source of complexity and diversity.

24.5 Confinement of Processing

SE-ODP runtime confines processing within capsules (or more strictly, the local operating system confines processing, and this is the SE-ODP way of modelling that and relating it to distribution transparency mechanisms).

In general terms, the main unit of confinement is usually a "program", consisting of modules created and brought together by the normal processes of program construction and linkage, loaded into a computer, and executed there as a distinct "process".

In SE-ODP terms, this confinement is modelled as a collection of processing components, modelled as pairs of interface adaptors and native components, executing in a capsule. This is shown in the object diagram in figure 23. There is one native component and interface adaptor pair per ODP component considered. Each of these native components has a connexion to its interface adaptor, which has a connexion to the nucleus (for continuity of description, these connexions are identified here with interactions A and B as in figure 19). The connexions from each object to the object that models the capsule are identified with some joint action (here labelled C) defining confinement constraints. This object diagram is not necessarily definitive: the confinement could be modelled in various other ways.

Figure 23 - The composition of a capsule

Because each of the native components and interface adaptors have a connexion to exactly one capsule, their action is confined there. But the nucleus also has
connexions (not shown in figure 19) to every other capsule considered. The nucleus is present at all of these capsules, but its action at each is confined there.

24.6 Spatial distribution

Every capsule considered has a connexion to an SE-ODP IPC object, via which the nucleus mechanizes any interactions between capsules. This is shown in the object diagram in figure 24. The nucleus is not visible in the diagram because it is considered here to be part of each capsule (as shown in figure 23).

![Figure 24 - Distributed processing](image)

A further level of detail can be added by considering location, which the SE-ODP models in terms of SE-ODP logical units. These are not necessarily in one-to-one correspondence with "physical units" visible in the technology viewpoint (see clause 17) - there may be multiple "ODP LUs" per "ODP PU".

The object diagram in figure 25 shows an example in which there are two locations. Location is modelled here by a connexion from each object to an SE-ODP logical unit. Each such connexion is identified with some joint action (here labelled L), defining location constraints. SE-ODP IPC is modelled here by an object at each location. This object diagram is not necessarily definitive: any such configuration could be modelled in various other ways.

![Figure 25 - Separate locations](image)

The connexions labelled X between capsules and SE-ODP IPC are the same as in figure 24 (i.e. figure 25 is a refinement of figure 24).
**SE-ODP logical units** are also a basic unit of modularity for managing the configuration of SE-ODP runtime, and they are points at which operational control can be exerted.

### 24.7 Viewpoints

As already stated, all of the structure of SE-ODP runtime described above is modelled in the engineering viewpoint.

Resources used by SE-ODP runtime mostly originate from the local operating system at each location, and are modelled in the technology viewpoint.

### 24.8 Applicability of Standardization

Not all of SE-ODP runtime needs to be standardized. Much of it can be left free to be implementation-specific.

To achieve interworking, the following must be standardized:

(a) the means of defining ODP interfaces (i.e. interface definition languages, for each of which there is a mapping to the Semantic Model);

(b) the specifications of the ODP interfaces (i.e. the abstract interfaces) of the particular application components considered;

(c) the externally visible services and protocols for SE-ODP IPC;

(d) the particular transfer syntax used in the relevant external interactions;

(e) ODP interfaces to the trading utility, and perhaps some of the other SE-ODP utilities.

For the above purpose, it is not necessary to standardize the application programming interfaces, or the interfaces between the native components and interface adaptors which are the realizations of ODP components, nor the interface(s) to the nucleus. These are local implementation matters. But some standardization of them may be needed to ensure correct mappings between them and the ODP-interface specifications (b) and for portability of application component implementations. This is for further study.

For procurement of different parts of SE-ODP runtime from different vendors, further standardization would be necessary.

Proposed SE-ODP standardization work items are considered in clauses 27-31.

### 25. SE-ODP MANAGEMENT ASPECT

#### 25.1 Introduction

This clause provides a preliminary description of management aspects of the SE-ODP.

#### 25.2 Managed Objects

A general principle is that all ODP components should be "managed objects" which would usually support ODP interfaces via which the ODP component can be integrated with management functions that are external to it.
The nature of these management functions is for further study. There would probably be standard ODP interfaces for generic management services which should be exported by ODP components. These services might include event reporting, meter monitoring, and basic operational controls of the ODP component (e.g. switch on, switch off, basic diagnostics and dump). There would also be application-specific management interfaces specific to whatever the individual ODP-component does (these would probably be derived by refinement of the generic management interfaces).

A major concern with such "managed objects" is the security implications of each object having rich management interfaces via which its behaviour might be observed and interfered with.

25.3 Management of SE-ODP Runtime

Management of SE-ODP runtime itself would be organized via distributed applications positioned as management utilities (see clause 24.4).

Various of these SE-ODP utilities would probably use the same data repository services as are used more generally for Computer Aided Software Engineering, CASE (see 8.9.6). This implies some degree of integration between CASE management and SE-ODP management.

25.4 Management of Distributed Applications

SE-ODP runtime would probably not have direct responsibility for managing the distributed applications which use it. But SE-ODP standards may require the constituent ODP components of distributed applications to have certain management characteristics (see 25.2 above), and SE-ODP runtime may be operated in ways that regulate use of applications (e.g. via control of trading).

25.5 Domain Structure

Trading operates with a domain structure which should be well aligned to the needs of system administration, configuration control and security (see Appendix D).

26. SE-ODP SECURITY ASPECT

26.1 Introduction

This clause provides a preliminary description of the security aspect of the SE-ODP.

26.2 Security and the RM-ODP

The general structure of the RM-ODP is intended to provide uniform and coherent descriptive modelling with near-universal applicability. The consequent normalized descriptions of systems should provide a well-formed basis for deployment of security architecture concepts.

The five viewpoints provide a systematic basis for relating together different kinds of social and technical decisions about security, and tying these back to the fundamental requirements of the enterprise, as expressed in the enterprise viewpoint. Furthermore, when the security requirements of the enterprise
change, their linkage to technical provisions in the other viewpoints should be visible. This is a basis for security design-analysis, design-control and audit.

26.3 ECMA Security Architecture

Current ECMA work on security architecture is documented in ECMA TR/46, which defines 10 Security Facilities, listed below.

- subject sponsor facility;
- authentication facility;
- association management facility;
- security state facility;
- security attributes management facility;
- authorization facility;
- interdomain facility;
- security audit facility;
- security recovery facility;
- cryptographic support facility;

The way in which the SE-ODP uses these Security Facilities is for future study. Most of them are likely to be positioned as SE-ODP security utilities (see 24.4).

26.4 Exploitation of SE-ODP Structure

Certain structural characteristics of the SE-ODP can be exploited for security purposes.

(a) The encapsulation and separation inherent in the capsule concept provides a basic granularity for security provisions.

(b) The ODP-association is the focus for intervention into object-access by Security Facilities.

(c) The SE-ODP IPC is a common mechanism through which all interactions between capsules must flow (no matter what diversity of networks etc. is involved). This is a key to unified assurance of security provisions.

(d) The SE-ODP interaction structure reveals finer grained application-specific structure which can be a basis for finer grained security controls (additional to those applied to capsules, the coarse-grained units).

The SE-ODP architecture does not make any a priori assumptions as to whether access is controlled by access-control-list, or capability mechanisms, or combinations of both. Further investigation may lead in any of these directions. What is certain is that cryptographic sealing of information will be widely necessary for authentication in distributed processing, and that this is also an enabling technology for distributed use of capability mechanisms.
26.5 Transfer of Security Information

ECMA TR/46 and subsequent TC32-TG9 work on security data elements have identified various requirements for passing security information between components of distributed systems. With the SE-ODP these exchanges would probably be mechanized via normal ODP-associations with appropriate security characteristics (e.g. using encryption).
SECTION FIVE - PROPOSED STANDARDIZATION
27. OVERVIEW OF PROPOSED STANDARDIZATION

27.1 Introduction
The clauses in Section Five propose future SE-ODP standardization work items. The descriptions are intended to be largely self-contained, and therefore they repeat some of what is already stated in other clauses.

This clause provides an overview and explanation of these standardization proposals.

27.2 Relationship to other Standardization
The SE-ODP is an area of standardization within Open Distributed Processing. This area spans the computation viewpoint and the engineering viewpoint of the RM-ODP, as illustrated in figure 26. The SE-ODP standardization defines more detailed structure within the outlines (to be) provided by the RM-ODP.

![Figure 26 - Relationship to RM-ODP](image)

SE-ODP standardization may affect and be affected by various other areas of standardization activity; e.g. OSI Upper Layer Architecture, CCITT Distributed Applications Framework (DAF), Remote Procedure Call, and the JTC1 study of Interfaces for Application Portability (IAP).

27.3 Emphasis
The proposed SE-ODP standardization recognizes the computation viewpoint as the primary focus for the SE-ODP.

The distinction relevant here is that the computation viewpoint is essentially about what the application programmer intends, and the engineering viewpoint is about supporting whatever is intended. The fundamental choices are those in the computation viewpoint which define application components and the interactions necessary between application components.

On this basis, the focus of SE-ODP standardization is on capturing the essence of distributed processing via interactions in the computation viewpoint, independent of the specifics of engineering and technological choices. Such structure is necessarily expressed in language, therefore this focus is linguistic. See figure 27.
Figure 27 - Linguistic Model

As SE-ODP standardization progresses, the emphasis will probably shift to the engineering viewpoint and to concerns about choices of "technology mappings" (i.e. how to use the technology that is modelled in the technology viewpoint).

27.4 Summary of Proposed SE-ODP Standardization

It is proposed that SE-ODP standardization should proceed in the following areas:

(a) Configuration (see clause 28);
(b) Interaction Semantics (see clause 29);
(c) Interface Definition Languages (see clause 30);
(d) Interconnection (see clause 31);
(e) Infrastructure Interfaces (see clause 32).

The crucial item for early start is (b), because it is not yet well understood, and all the other areas are to various degrees dependent on it. But work on area (a) should also be started immediately, because it pervades the whole architectural structure of the SE-ODP.

28. SE-ODP CONFIGURATION STANDARD

28.1 Introduction

This clause proposes an area of SE-ODP standardization that is referred to as the "SE-ODP Configuration Standard".

This subject has been recognized and explained in clauses 18, 22 and 24, and in Appendix D.

The area of standardization is outlined here by briefly stating its scope (see 28.2) and its purpose and justification (see 28.3).

28.2 Scope

Techniques collectively termed "trading" are the main means of configuring the SE-ODP infrastructure and distributed applications using it. This SE-ODP standardization work will:

(a) define the SE-ODP Trading Model;
(b) apply the SE-ODP Trading Model in each of the five viewpoints of the RM-ODP;
(c) evaluate this \textit{SE-ODP Trading Model} by applying it to the X.500 Directory System;

(d) evaluate this \textit{SE-ODP Trading Model} by applying it to the ECMA TR/46 Security Framework (and progressions of TR/46);

(e) integrate the SE-ODP Trading Model with Computer Aided Software Engineering (CASE) provisions.

(f) define the supportive \textit{services} provided by the \textit{SE-ODP Trading Service} and;

(g) specify the \textit{distributed application} that provides this \textit{Trading Service}.

\subsection*{28.3 Purpose and Justification}

This Configuration standard is fundamental to \textit{SE-ODP} structure.

The work of specifying \textit{trading} should flush out and resolve key issues concerning naming, binding, typing / subtyping, configuration control, access-control, security, management, federation, scaling and software engineering.

\section*{29. SE-ODP INTERACTION SEMANTICS STANDARD}

\subsection*{29.1 Introduction}

This clause proposes an area of \textit{SE-ODP} standardization that is referred to as the "\textit{SE-ODP Interaction Semantics Standard}".

This subject has been recognized and explained in clause 18.

The area of standardization is outlined here by briefly stating its scope (see 29.2) and its purpose and justification (see 29.3).

\subsection*{29.2 Scope}

This \textit{SE-ODP} standardization work will define computational semantics common to all \textit{interactions} in \textit{Open Distributed Processing}. The resultant Semantic Model will be applicable to all "interface definition languages" in which external \textit{interactions} are specified and implemented (see clause 30).

Existing semantic models of computation (e.g. von Neumann architectures) were developed to describe computing in non-concurrent, single-processor environments. Languages based on such semantic models do not adequately express all concepts applicable to \textit{distributed processing}, such as distribution, concurrency and encapsulation.

The abstractions in this Semantic Model will determine the set of all possible \textit{interactions} that conform to ODP Standards and are valid in the \textit{computation viewpoint}. The Semantic Model thereby provides criteria against which to validate the adequacy and correctness of \textit{SE-ODP} provisions. Similarly, it will help to reveal whether a technical choice in the \textit{engineering viewpoint} is about different ways of doing the same thing, or is about fundamentals.

The semantics will be expressed in formal (i.e. mathematical) notation. But an interface definition language conforming to these semantics (see clause 30) is not
restricted to mathematical notation (the essential constraint is the existence of an agreed mapping of the language onto the Semantic Model).

The standard will also define conformance criteria and process via which languages/notations can be qualified as "interface definition languages" consistent with these interaction semantics. See 30.2 (a) for an example of this qualification process.

29.3 Purpose and Justification

The Semantic Model to be defined in this standardization is fundamental to the architectural structure of the SE-ODP. Without a definitive and comprehensive Semantic Model, the unambiguous and consistent interpretation of the semantics of interactions is impossible.

30. SE-ODP INTERFACE DEFINITION LANGUAGES STANDARD

30.1 Introduction

This clause proposes an area of SE-ODP standardization that is referred to as the "SE-ODP Interface Definition Languages Standard".

This subject has been recognized and explained in clause 19.

The area of standardization is outlined here by briefly stating its scope (see 30.2) and its purpose and justification (see 30.3).

30.2 Scope

The subject is standardization of languages for definition of SE-ODP interfaces which define and constrain the interactions between application components of distributed applications that use the SE-ODP. The main requirement against which to evaluate such languages is their conformance with the Semantic Model (see clause 29).

This SE-ODP standardization work will:

(a) evaluate and qualify for use as "interface definition languages" selected existing languages / notations such as ISO 8824 ASN.1, ISO 9072 Remote Operations notation, ECMA 127 RPC notation and ISO 8807 LOTOS;

(b) define a machine-processable language specialized for definition of ODP-interfaces;

(c) explore whether there is a need for standardization of programming languages specialized for this kind of remote interaction.

The existing languages / notations (a) were not designed to conform to the particular requirements of the Semantic Model (e.g. the semantics of ISO 9072 Remote Operations are specified by informal connotation, not machine-processable denotation). We expect that none of these languages will express all of the concepts etc. defined in the Semantic Model.

The specialized language (b) for definition of SE-ODP interfaces will entirely conform to the requirements of the Semantic Model. It therefore necessarily includes machine-processable denotation for the signatures, semantics, and
relevant distribution transparencies of the interactions. This completeness may lead towards wider applicability as a language for implementing interactions that occur as realizations of SE-ODP interfaces. Hence the investigation (c).

30.3 Purpose and Justification

Enabling the use of existing languages for the definition of SE-ODP interfaces, per (a) above, is essential for continuity of investment and for phased introduction of SE-ODP techniques.

The specialized Interface Definition Language for the complete definition of interactions, (b) above, is a vital ingredient of SE-ODP standardization. It is the general means for implementation-independent definition of interactions between application components. It is a natural focus for investment in software engineering tools to support development of computer applications that use the SE-ODP.

31. SE-ODP INTERCONNECTION STANDARD

31.1 Introduction

This clause proposes an area of SE-ODP standardization that is referred to as the "SE-ODP Interconnection Standard".

This subject has been recognized and explained in clause 23.

The area of standardization is outlined here by briefly stating its scope (see 31.2) and its purpose and justification (see 31.3).

31.2 Scope

An interaction defined between application components in the computation viewpoint encounters three different sets of circumstances in the engineering viewpoint. It may occur:

(a) internally to what is termed a capsule (defined in clause 19);
(b) between capsules that are physically co-located;
(c) between capsules that are in physically separate locations.

We are concerned here with standardization for the cases where the application components are in separate capsules; i.e. (b) and (c) above. It should be noted that the semantics of the interaction are the same in all three cases (a), (b), (c); only the engineering and technology are different.

This standardization work will:

(d) define the interactions between SE-ODP capsules (i.e. the requirement to be satisfied by protocols etc.);
(e) define mappings of these interactions onto existing OSI services (with consequent re-use of existing OSI protocol stacks);
(f) explore other interconnection services and protocols to support characteristics required for (d); e.g. for isochronous interactions.
Item (d) may require specification of the **SE-ODP processing model** (see clause 19) to an appropriate level of detail, so that the requirement can be expressed in terms of the **services** of **SE-ODP IPC**.

The specialized "interface definition language" (see clause 30) might be used for (d). The specifications (e) will include provisions for compatibility with ECMA-127 RPC interconnection.

### 31.3 Purpose and Justification

External interconnection is an area in which standards are necessary to ensure compatibility and successful interworking across communications networks.

### 32. SE-ODP INFRASTRUCTURE INTERFACES STANDARD

#### 32.1 Introduction

This clause proposes an area of **SE-ODP** standardization that is referred to as the "**SE-ODP Infrastructure Interfaces Standard". This subject has been recognized and explained in clause 24.

The area of standardization is outlined here by briefly stating its scope (see 32.2) and its purpose and justification (see 32.3).

#### 32.2 Scope

The subject of this standardization is interfaces of the SE-ODP infrastructure (termed **SE-ODP runtime**) that supports **interactions** via the **SE-ODP**. These interfaces would normally be visible to compilers and interpreters, but not to **application programmers**. These infrastructure interfaces are consequently defined as **abstract interfaces** without specification of language-bindings for them.

These infrastructure interfaces are local implementation matters, and standardization of them is not strictly necessary for interworking compatibility and application portability. This is because interworking compatibility and portability of applications software using the **SE-ODP** can, in principle, be assured by the ability of software tools to adapt programs to use arbitrary support environment interfaces (e.g. implementation-dependent interfaces). However, there is a case for variety reduction (see 32.3).

This **SE-ODP** standardization work will, as necessary:

(a) define the "dynamic binding service" to be provided by **SE-ODP run-time** (this uses, and is closely related to, the **trading service** identified in 28.2);

(b) define the "invocation service" to be provided for **interactions** via the bindings (a), by **SE-ODP run-time**; and

(c) define a software interface via which the combined services (a) and (b) are provided and used.

The specialized interface definition language identified in clause 30 might be used.
32.3 Purpose and Justification

This standardization is important because of the simplifications, saving and opportunities for multi-vendor procurement that can result from reducing the variability of support environment interfaces.

Other standards activities such as Posix are defining application programming interfaces that may be functionally similar to these SE-ODP infrastructure interfaces. The former should be taken into account by this work item, although they are language-dependent concrete interfaces visible to application programmers (unlike the abstract interfaces considered here, which are not intended to be visible to application programmers).
APPENDICES
APPENDIX A

BIBLIOGRAPHY

*ODP* standardization should make full use of the known results of research and practical experience.

There is no single work of reference that provides a complete survey of all the progress that has occurred in the field of *distributed information systems*. The references below have been selected to provide a basic reading list. These sources have been variously used in ECMA *ODP* work, and in the *SE-ODP* prototyping activities referred to in 7.5. They are gratefully acknowledge.

A previous version of this reading list has been contributed by ECMA to the ISO *ODP* work (as JTC1 SC21/WG1/N363), together with a tutorial survey of *distributed processing*.

[ACCETTA 86]


[AGUILAR 86]


[ANSA 89]


[BIRMAN 85]


[BIRMAN 86]


[BIRMAN 87]

[BIRRELL 82]

[BIRRELL 84]

[BIRRELL 85]

[BIRRELL 86]

[BLACK 85]

[BLACK 87]

[BROWNBRIDGE 82]

[CARDELLI 85]

[CHECKLAND 86]

[CHERITON 84]

[CLARK 86]
[COOPER 85]

[DELTA-4 88]

[DOD 80]

[FORSICK 85]

[GRAY 79]

[GIBBONS 87]

[GOLDBERG 83]

[JONES 85]

[JONES 79]

[KRUEGER 88]

[KUNG 81]
[LAMPSON 83]

[LAPRIE 85]

[LESLIE 84]

[LISKOV 82]
Liskov, B. & Scheifer, R., "Guardians and Actions: Linguistic Support for Robust Distributed Programs", ACM Transactions on Programming Languages and Systems, 5(3), 381-404 (July 1983).

[MITCHELL 82]

[MORRIS 86]

[MULLENDER 86]

[NEEDHAM 78]

[NEEDHAM 82]

[POPEK 81]
[RASHID 81]

[REMER 75]

[ROBINSON 88]

[RUSHBY 83]

[SALTZER 84]

[ SCHROEDER 84]

[SPECTOR 85]

[SVOBODOVA 84]

[TANENBAUM 85]

[VOYDOCK 83]

[WALKER 83]
[WEGNER 87]

[WIRTH 83]
APPENDIX B

BASIC CONCEPTS

B.1 GENERAL

B.1.1 Scope

This Appendix presents ODP Terminology, concepts and architectural structure that are used in the main body of this Technical Report, on the assumption that they will be defined in the Reference Model of Open Distributed Processing (RM-ODP).

This Appendix is a consolidation of and progression from current ISO working papers and ECMA contributions to production of the RM-ODP in ISO.

The RM-ODP is concerned with architecture that is intended to be applicable to most kinds of application. Other standards work items are concerned with domain-specific architectures that are not intended to have this general applicability; e.g. the Distributed Office Applications Model (DOAM) in SC18. In the mature phase of ODP standardization, these domain-specific architectures should each be consistent with the RM-ODP.

This relationship to domain-specific architectures is illustrated in figure 28.

![Figure 28 - Relationship to Domain-specific Architectures](image-url)
B.1.2 Basic Definitions

The following definitions apply.

**processing**: programmed *activity* executed in computers.

**distributed processing**: *processing* that may span separate computer address spaces.

**Reference Model of Open Distributed Processing (RM-ODP)**: a reference model to be defined by ISO/IEC JTC1 SC21.

**open distributed processing** (ODP): *distributed processing* conforming to requirements specified in the **RM-ODP**.

All other definitions of terms in this Appendix are included at the beginning of individual clauses and subclauses.

As an aid to understanding the structure of the definitions etc., references to them are usually in *italics*.

B.1.3 Overview

Basic *systems* concepts and terminology are presented in B.2.

The general applicability of *distributed systems* is described in B.3.

*Distribution transparencies*, which are considered to be a distinctive ingredient of *distributed processing*, are presented in B.4.

The Framework of Abstractions used in the RM-ODP is presented in B.5.

*Object* concepts, which are the basis for the general modelling technique used in ODP, are presented in B.6.

B.2 SYSTEMS

B.2.1 General

This subclause presents a very general approach to the subject of "*systems*" and provides the context for considering *distributed systems*.

B.2.2 Definitions

The following definitions apply:

**system**: a composite whole.

**component**: any item that contributes to the composition of some whole.

**subsystem**: a *system* considered as a *component* of some other *system*.

**information system**: any *system* that processes, or stores, or transfers information.

**social system**: a *system* that is considered to include human activity.

**technical system**: a *system* that is man made and is not a *social system*.

**socio-technical system**: a *social system*, some *subsystem* of which is explicitly considered to be a **technical system**.
natural system: any system that is neither a social system nor a technical system.

information technology system: a technical system that is an information system.

IT system: abbreviation for information technology system.

distributed information system: an information system, some components of which are considered to be separate.

distributed IT system: a technical system that is a distributed information system.

distributed system: abbreviation for distributed IT system where this meaning is clear from the context.

computer application: productive activity exerted via computers.

application program: the autonomously executable specification of (part of) a computer application.

application component: an application program considered in terms of its contribution to the composition of some computer application whole.

distributed application: a computer application composed of discrete application components.

B.2.3 Information Systems

The same information system may be a social system or a technical system, depending on which components are considered.

For example, a typical accounting system is an information system consisting of the accounts department, the people working in it, and the computer databases and applications programs used for the accounting functions. Considered as a whole it is a social system because it includes human activity. But considered only in terms of the computers, databases and software, it is a technical system (specifically an IT system). Both ways of viewing it are valid and necessary. The term socio-technical system explicitly recognizes this duality which is inherent in most systems that are of interest to us here.

All information systems have some interactions with natural systems (e.g. their physical environment). Some information systems may be designed specifically to have interactions with a natural system (e.g. a weather forecasting system interacts with weather systems).

ODP standardization is only concerned with standards for IT systems. That is why the distinction is made here between technical systems and social systems. This distinction is not always explicitly made (e.g. see figures 30 and 31).

B.3 APPLICABILITY OF DISTRIBUTED SYSTEMS

As a consequence of general organizational, logistical, historical and technical constraints arising from the real world, IT systems tend to be dispersed into many separate programs which need to be integrated together as distributed systems. See figure 29.
Figure 29 - Integration of separate programs

Therefore, the field of application of distributed IT systems and distributed applications is virtually unlimited and very diverse. It includes, but is not limited to, such areas as:

(a) Data Processing Systems;
(b) Office Systems;
(c) Process Control Systems;
(d) Command and Control Systems; and
(e) Integrated Data/Text/Voice/Image Systems.

Distributed systems are important not only within each field of application, but also as a bridge between different fields of applications. They can be the means of integrating separate functions into a large scale information system, see Figure 30.

Figure 30 - Integration of separate information systems within an organization

Such co-ordination and cooperation is also necessary between separate organizations. This kind of integration of autonomous information systems is another case of the general applicability of distributed systems, see figure 31.

Figure 31 - Integration of systems in separate organizations
Wide applicability of ODP standardization therefore arises from the wide applicability of distributed systems and distributed applications.

There are some finer distinctions between networking, network operating systems and distributed operating systems that have not yet been made in the RM-ODP work, but are relevant to SE-ODP standardization. See Appendix C.

B.4 DISTRIBUTION TRANSPARENCIES

B.4.1 General

This subclause defines and briefly explains distribution transparency.

The separation, or dispersal, of the components of a distributed IT system requires that there is explicit communication between interacting components. Furthermore, this separation allows truly parallel execution of programs, the containment of independent faults to the failing components, and the use of separation as a means for enforcing security policies. It also allows incremental growth and contraction of the system.

A key issue in the design of a distributed IT system is how, and to what extent, to conceal the complexities of the distributed nature of the system, while exploiting some of the effects of separation to obtain desirable qualities.

This is handled by the selective use of the various mechanisms for distribution transparency.

B.4.2 Definitions

The following definitions apply.

distribution transparency: means of hiding consequences of distribution from distributed applications and their human users.

access transparency: a distribution transparency which hides the mechanisms used to achieve interaction.

location transparency: a distribution transparency which hides the location of interacting components of a distributed system.

concurrency transparency: a distribution transparency which hides simultaneous access to shared resources.

replication transparency: a distribution transparency which hides replication of a component into multiple separate instances.

failure transparency: a distribution transparency which hides the effects of a failure of a component of a distributed system.

NOTE B.1:
According to the definitions in [LAPRIE 85] "failure transparency" might more correctly be termed "fault transparency" ... hiding ... faults.

migration transparency: a distribution transparency which hides the effects of relocating a component of a distributed system.
B.4.3 The Transparencies

A distributed IT system may be said to have distribution transparency if it provides means of hiding the effects of distribution from the components and human users of the system.

There are a number of specializations of distribution transparency which require different transparency mechanisms.

Access transparency allows invocations on objects irrespective of how the objects are accessed, and irrespective of whether the objects are co-located or remote. It requires generic functions for communications, for security and for related management.

Location transparency ensures that the bindings between objects are independent of the routes that connect them. It requires functions for naming, addressing and routeing.

Concurrency transparency allows the parallel use of a resource without permitting inconsistent use of that resource. It requires functions for event ordering, synchronization, mutual exclusion and support for atomic operations (transactions).

Replication transparency allows multiple instances of objects to be used to increase dependability and performance. This requires functions for both active replication (where all the instances are simultaneously in operation) and passive replication (where one instance operates and the others provide stand-by).

Failure transparency enables the full concealment of faults despite the failure of components. It requires functions for fault management and fault recovery.

Migration transparency allows the movement of an object in a distributed system without making the transition apparent to interacting objects. It requires functions for object management, and support for dynamic re-configuration.

B.4.4 Selective Transparency

Complete distribution transparency, while concealing all the complexities that result from component separation, may deny the designer of a distributed system the opportunity to exploit some of the advantages of separation. Furthermore, the mechanisms for unwanted distribution transparency may exact an unacceptable performance overhead, particularly in real-time systems.

For these reasons it is desirable that a support environment for distributed processing systems should provide for selective use of the transparencies.

B.5 FRAMEWORK OF ABSTRACTIONS

B.5.1 General

This clause summarizes the "Framework of Abstractions" around which the RM-ODP is to be structured.
B.5.2 Definitions

The following definitions apply:

**Viewpoint**: a model in which an information system is viewed in terms of a particular set of concerns.

**Enterprise viewpoint**: a viewpoint for modelling what the information system is required to do.

**Information viewpoint**: a viewpoint for modelling the information structure of the information system.

**Computation viewpoint**: a viewpoint for modelling the algorithmic structure of the information system.

**Engineering viewpoint**: a viewpoint for modelling how qualitative characteristics of the information system are assured.

**Technology viewpoint**: a viewpoint for modelling the realized components from which the information system is constructed.

**Aspect**: a subject area that pervades all five viewpoints of the RM-ODP.

B.5.3 Viewpoints

In order to avoid dealing with the full complexity of a distributed system all at once, the system may be considered from different viewpoints, each presenting a different abstraction of it.

Each viewpoint addresses a particular concern. It does so by fully recognising the characteristics of the system that are relevant to that concern and simplifying all others. The latter are still present in the viewpoint, but are less precisely expressed and are possibly merged with others.

A number of viewpoints are possible, but a set of five viewpoints has been selected. This is a complete and reasonably minimum set, and the chosen viewpoints are considered to be those most useful in the description, analysis and synthesis of complex distributed IT systems.

The framework of abstractions organises these five viewpoints and allows complete description of the system, consistent across the differing viewpoints.

The names of the selected viewpoints, and the concerns that they address, are as follows.

- **Enterprise viewpoint**:
  
  Human and social issues.
  Management and finance.
  Legal concerns.

- **Information viewpoint**:
  
  Information modelling, flow and structure.
- **Computation viewpoint:**
  Application design and development.
  (Algorithms for concurrent and distributed computing)

- **Engineering viewpoint:**
  Infrastructure.
  Application support.
  Transparencies, naming, binding.

- **Technology viewpoint:**
  Constraints imposed by technology.

The *enterprise viewpoint* describes the *information system* in terms of what it is required to do. The *model* from this viewpoint captures the business and administrative requirements and policies that justify and orientate the design of the *system*.

The *information viewpoint* describes the *information system* in terms of the *information structure*, *information flow* and *information manipulation constraints*.

The *computation viewpoint* describes the *information system* in terms of the operation and computational characteristics of the processes that change the *information*. It concerns the structuring of applications so that they are independent of the underlying computers and networks.

The *engineering viewpoint* describes the *information system* in terms of the engineering necessary to support the distributed nature of the *processing*. It is concerned with the provisions and assurance of desired characteristics such as performance, dependability and *distribution transparency*.

The *technology viewpoint* describes the *information system* in terms of the realised *components* from which it is built. E.g. it models the hardware and software of the local *operating systems*, the input/output devices, storage and communications.

While each of the *viewpoints* describes only those matters that are relevant to its concerns, some matters will be common to two or more *viewpoints*. In particular the *information viewpoint* and *computation viewpoint* have many matters in common.

### B.5.4 Aspects

A *distributed processing system* may be considered from certain *aspects* which pervade system structure. The aspects identified are: Storage, Process, Distribution, Identification, Management, Security and User Access. These concerns generally pervade the five *viewpoints*, and structuring of them is for further study.

### B.5.5 Levels of abstraction

Each *viewpoint* is a complete description of the *information system*, concentrating on the concerns of that *viewpoint*. The description may be
specified in narrative form, or using some formal technique, or a combination
of narrative and formal specification.

Any such description may be changed by altering the level of abstraction, i.e.
by either adding or removing details (more strictly, these are the processes of
composition and decomposition by refinement).

A description at a high level of abstraction gives a broad overview of the
system from that viewpoint. At a lower level of abstraction the same system is
described from the same viewpoint but with more detail.

B.6 OBJECT MODEL

NOTE B.2:
This subclause is a progression beyond the ISO ODP work current at the time of drafting the

B.6.1 General

B.6.1.1 Introduction

This subclause defines the object model used in the Technical Report.

The RM-ODP architectural structure is to be described via one object
model, used throughout the five viewpoints defined in the framework of
abstractions (see B.5); i.e. one and the same object model for architectural
modelling in the enterprise viewpoint, information viewpoint, computation
viewpoint, engineering viewpoint and technology viewpoint.

At the time of drafting this Technical Report there is no agreed object
model for this purpose in ISO. A minimal object model is tentatively
outlined in this clause in order to provide a consistent basis for modelling
in the main body of the Technical Report. The criteria when formulating it
were simplicity, and maximum coverage with the minimum number of
concepts.

This is a simplified and generalized subset of what is termed termed "object
engineering" in [ANSA 89]. The latter includes further levels of detail that
have not yet been studied in ECMA (e.g. notations for defining actions and
the constraints upon them, class / type concepts and inheritance concepts;
and specification of the formal mathematical basis of the object model).

B.6.1.2 What to Model?

The model outlined in this clause is intended for description of, reasoning
about, and manipulation of, the structure of discrete information systems
composed of discrete components (i.e. the modelling of structure as distinct
from content). A system is to be modelled as an object, and each component
is to be modelled as an object.

For this modelling of system structure it is necessary and sufficient to
model:

(a) the existence and identification of the discrete objects;
(b) the constraints applicable to objects in combination;
the rules of composition for objects in combination.

A complete model of system structure (a), (b), (c) is not a fully refined specification of the objects and the interactions between them (but is none the less complete in itself). The specification of the objects and interactions themselves is another matter. It would require further (and often voluminous) detail which is not relevant to this purpose; and which would by its presence obscure the essentials of structure.

This object model is therefore necessarily such that detailed specification of objects and interactions can be added by refinement. As a practical matter, the specifications of system structure, objects and interactions would usually be separate items, with appropriate cross-referencing between them.

This is analogous with the way in which electronics engineers use circuit diagrams and component data sheets. The required object model is for use in "circuit diagrams" of system structure. The "data sheets" of components (objects) and their interactions are a separate matter.

Furthermore, different kinds of specification technique may be applicable to different kinds of objects and interactions in different viewpoints. This is further justification for making the choice of techniques for specification of objects and interactions a matter separate from specification of system structure.

This one object model for ODP architecture does not preclude use of object models for other purposes (e.g. component specification, design and programming). These other purpose may be served by other object models (and perhaps by re-use of this general object model).

B.6.2 Terminology

B.6.2.1 Definitions

The following definitions apply:

activity: the exertion of influence.

action: a unit of activity.

object: that to which activity is attributed.

joint action: action attributed to a pair of objects.

connexion: a model of constraints on possible joint action.

interaction: joint action in which action attributed to one object constrains action attributed to the other.

model: a representation of some subject.

object model: a model in which objects are represented.

object diagram: an object model in the form of a diagram.
B.6.2.2 Explanation of Terminology

B.6.2.2.1 Activity

These definitions are bootstrapped from the general concept of "activity", for which a very general definition is used.

For theoretical reasons it may be necessary to underpin these definitions with more primitive concepts of "substance", "occurrence", etc. This is for further study.

B.6.2.2.2 Object

The term "object" is in widespread use, often in conjunction with various other terms (e.g. "object-oriented"). It is overloaded with diverse meanings (e.g. every-day meanings and programming language meanings).

The definition used here is intended to be sufficiently general to be applicable to all of the rather different kinds of "objects" visible throughout the framework of abstractions defined in B.5. It rests on the minimum assumption that an object will at least "exert influence". Any "object" that did less could hardly be of interest to ODP.

B.6.2.2.3 Joint action and Interaction

The definition of joint action is intended to be sufficiently general to be applicable throughout the framework of abstractions defined in B.5.

Interaction is a special case of joint action; i.e. there can be joint actions which are not interactions. Therefore, the more general concept of joint action is used to construct this basic object model (although in practice most joint actions of interest are interactions).

Another reason for describing the model in terms of constraints on joint action (instead of constraints on interaction) is to help to make clear that the specifying of interactions is not itself the purpose of this object model.

B.6.2.2.4 Connexion

The term "connection" is in widespread use. It is overloaded with diverse meanings which are applicable in various contexts (e.g. the particular meaning used in OSI standardization); but such meanings are not necessarily applicable throughout the RM-ODP.

What we are modelling here is "connection" as a constraint on joint action. This does not have the same connotation as "connection" in telecommunications; i.e. it does not necessarily imply the existence of a communications channel.

The alternative spelling "connexion" is used to distinguish the particular meaning defined in B.6.2.1. The definition is intended to be sufficiently general to be applicable throughout the framework of abstractions defined in B.5.
B.6.2.2.5 Origins

These definitions originate from a history of scientific debate about the nature of "objects", conducted over the Centuries. An early formulation of this kind of general object model was by the 17th Century German mathematician Leibnitz. His model attributed all exertion of influence to autonomous indivisible objects (which he called "monads") between which there was joint action, and constraints upon joint action (i.e. connexions), but no interaction whatsoever (which was an implausible restriction).

The "object-oriented paradigm" of computer science is a recent invention, specialized for computer programming, etc.

B.6.3 Model

B.6.3.1 General

The purpose, terminology and tutorial introduction to this object model have been given above. The model and its notation are now briefly described.

B.6.3.2 Concepts

An object is specified in terms of the joint actions in which it is considered to participate. These joint actions are represented by connexions. An object may have an arbitrary number of connexions, to the same or different objects, including itself.

This object model has no way of describing the activity of an object which has no connexions.

The objects, connexions and joint actions in this object model are models; i.e. they themselves are not what actually occurs. At most they are an accurate description of what can occur.

The simplification that any connexion is always between exactly two objects is not restrictive. Arbitrarily complex connectivity can be modelled by multiple connexions (e.g. by fan-out via intermediate objects). Also for a connexion that is not fully determined, an object may represent "any object", or "the rest of the system".

Further study is needed to define these concepts in more detail. In particular, equivalence rules are needed (and consequent transforms); e.g. equivalence between a connexion identified with a compound joint action, and some combination of connexions each of which is identified with a more elementary joint action which is also present in the compound joint action.

B.6.3.3 Object Diagram Notation

An object diagram notation is used for this object model. The choice of notation is independent of the choice of concepts. This particular notation is oriented towards white boards and explaining how to build things.
An object is represented by a circle, with an adjacent textual label that provides identification. The circle shall not overlap a circle representing any other object. The size of the circle, its position in the diagram and the thickness of the line used in drawing it have no logical significance.

A connexion is represented by an arc. The thickness and orientation of the line used in drawing it have no logical significance. If known, the joint action with which a connexion is identified is shown by an adjacent label.

Arcs may cross one another (and the crossing has no logical significance); but arcs may not cross circles (objects). Each end of an arc touches a circle (object).

An object diagram is a connected graph in the graph theoretic sense (and therefore shall have no dangling arcs, and no unconnected nodes).

B.6.4 Using the Model

B.6.4.1 Examples

Figure 32 illustrates some examples. Example 1 shows an object connected to a collection of unknown objects (i.e. an object representing the rest of the system) and in unknown ways (i.e. no particular joint action is identified with the connexion). Example 2 illustrates two objects with a single connexion between them. In example 3 there are multiple connexions between two objects. In example 4 there is connexion fan-out to multiple objects.

![Object diagram examples](image-url)
These examples illustrate the wide applicability of this object model and its graphical representation. Example 1 is typical of an incomplete design at an exploratory stage. It might be a predecessor of the more detailed diagrams in any of the other examples. Example 2 might be a less detailed view of example 3 or example 4. From the labels it is apparent that example 3 is a model of (part of) an information system in which client programs have three different kinds of joint action with a file service (X might be file access, Y file transfer, and Z file management). Organizational roles are modelled in example 4 (not computer software).

Throughout this Technical Report most of the illustrations are object diagrams conforming to this object model.

B.6.4.2 Information Hiding

An essential characteristic of object models is that an object models what is possible, not how it is made possible. The "how" (i.e. the way in which the object is realized) is hidden, encapsulated within the object. This allows specification to be reduced to bare necessities, and to be kept independent of more detailed levels of design and attendant implementation choices.

In this object model the composition of an object can only be revealed as a more detailed arrangement of objects with connexions to the object considered; i.e. detail is added by refinement. In the example illustrated in figure 33, some "file server" object is decomposed into a possible more detailed arrangement of objects not previously visible. The diagrams in figure 33 are transformations of one another.

![Diagram](image)

**Figure 33 - Example of object decompositon**

Such information hiding is the key to incremental formulation and description of a design. Each increment (i.e. the revealing of more objects, connexions and details of joint actions) is a refinement of its predecessor.
(and not a reformulation). This incremental process can be started without necessarily knowing or anticipating the final structure.

In an object diagram all that is known about an object is the lines which wrap around it and mark out its connexions, and the labels that accompany the various lines. Most of an object diagram is white space. The white space encapsulated within each object symbolizes its unknown content. This is totally separated from the white space all around the objects, which symbolizes all things outside the universe of discourse.

B.6.5 Formal Description

Definition of the formal (i.e. mathematical) basis of this object model and of the object diagram notation is for further study. But some preliminary observations can be made.

The most important observation is that providing a formal basis for this model of system structure is a matter separable from detailed formal description of objects and interactions.

The formal basis for description of such system structure in [ANSA 89] is graph theoretic arrangement of objects, and set theoretic description of constraints. For reasons of brevity and general applicability, the constraints on joint actions are symbolized at a high level of abstraction, with a consequent coarse level of detail. The technique is oriented towards achieving the necessary degree of accuracy (i.e. formalism) with minimum precision (i.e. avoiding unwanted degrees of detail). This accuracy/precision trade-off invites selective use of more specialized formalisms (e.g. abstract data types, process algebras and set theoretic inheritance hierarchies) to provide higher precision where needed. E.g. a symbol representing some joint action in an object diagram could be expanded into the specification of a set of events and partial orderings, defined in ISO 8807 LOTOS notation.

It is likely that LOTOS will be used for detailed ODP specifications of objects and interactions.
APPENDIX C

DISTRIBUTED SYSTEM CONCEPTS

C.1 GENERAL
This Appendix considers different kinds of approach to distributed systems, and introduces the terms networking, network operating system and distributed operating system. These distinctions may be needed in the RM-ODP.

C.2 DEFINITIONS
The following definitions apply.

**networking**: use of remote resources accessed via a communications network.

**networking facilities**: facilities that support networking.

**operating system**: an IT system that organizes resources pertinent to IT systems.

**network operating system**: an operating system that is itself a distributed IT system and organizes resources for remote use.

**distributed operating system**: an operating system that is itself a distributed IT system and organizes resources for local and remote use.

C.3 NETWORKING
The essence of networking is that the resources accessed are explicitly considered to be remote. (They may be in close proximity or geographically distant.)

Networking facilities are concerned with matters such as terminal access, remote log on, remote job entry, file transfer, remote file access, remote program access, electronic mail and remote printing; and with provision of the interconnection that is an ingredient of all these. There may also be networking facilities concerned with managing remote resources.

The resources accessed are usually organized by autonomous local operating systems. The networking may therefore span across multiple separate operating systems. These operating systems may be heterogeneous, and the networking facilities that are superimposed on them may provide some degree of homogeneity to the remote users of the resources.

Opportunities for shared use of resources are usually an important characteristic of networking.

A networking facility is usually provided in ways such that it itself is a distributed IT system.

C.4 NETWORK OPERATING SYSTEM
A network operating system provides a collection of networking facilities that organize the access to remote resources in a coherent way.
Organizing local use of resources is outside the scope of a network operating system, and is a matter for a local host operating system. Therefore, any network operating system necessarily co-exists with the host operating system(s). It might, or might not, exercise some degree of hierarchic control over the host operating system(s).

C.5 DISTRIBUTED OPERATING SYSTEM

A distributed operating system, as defined here, is different from a network operating system, in that it supports access to local and remote resources. Not being restricted to remote access, its services are potentially as comprehensive as those of any operating system.

In principle, a distributed operating system may organize all of the relevant resources; no other operating system need be present. In practice, a distributed operating system may be hosted by local operating systems.

A distributed operating system provides homogeneous services to its users. (This does not preclude the latter from using the services of any other operating systems that are accessible to them). The computers and interconnection across which a distributed operating system operates may be homogeneous or heterogeneous.

The distinction between the concepts of distributed operating system and network operating system is not absolute, and these terms have been used with different meanings in different circumstances.

C.6 DISTRIBUTED APPLICATIONS

The main use of distributed processing is for realization of distributed applications.

But distributed processing is not the only way in which distributed applications can be realized. Some alternatives are:

(a) Networking facilities such as file transfer, electronic mail and remote file access, may be used to construct some kinds of distributed applications.

(b) Some kinds of distributed applications can be constructed by using distributed database and distributed virtual memory (although these themselves usually depend on an underlying specialized distributed operating system and its distributed processing, which they mask).

Distributed processing is something that networking and network operating systems and distributed operating systems can all support, to varying degrees. But they all also support other things (e.g. they may support file store and job control).
D.1 GENERAL

In the main body of this Technical Report the need for a Trading Application is identified, and a future work item is proposed to develop a standard defining such a model for SE-ODP purposes (see clause 28).

This appendix is a preliminary description of trading concepts which have been contributed to ECMA as a proposed basis for this future work. It originates from the ANSA project [ANSA 89]. At this stage, the detail in this appendix is not fully consistent with the main body of the Technical Report.

Trading establishes a relationship between objects which permits interaction between them. The Trading Application is used by SE-ODP runtime, as explained in clause 24.

D.2 DEFINITIONS

The following definitions apply.

specification: a predicate which can be applied to items in a domain of discourse.

instance: an item in the domain of discourse which satisfies a particular specification.

class: the set of all items in the domain of discourse satisfying a particular specification.

type: the specification of a class.

import: a proposal to use some particular service.

export: a proposal to provide some particular service.

importer: an object to which the use of a service is attributed.

exporter: an object to which provision of a service is attributed.

interface specification: a description of the required behaviour of the importer and exporter when engaged in interactions.

association: the relationship between an importer and exporter that allows the importer to invoke operations on the exporter. It is based upon an interface specification for the service in question. (This is not an OSI layer 7 association.)

trading: activity pertaining to imports and exports.

trading service: a service used to match imports and exports.

trading system: an implementation of the trading service.

trader: an object to which provision of a trading service is attributed.
trading context: within a trading system, a naming context in which an export is registered.

trading scope: the set of trading contexts reachable from a given trading context, including the initial context.

trading domain: the logical scope of a trading system.

federated trading systems: cooperation of two or more trading systems to provide a trading service spanning the trading systems.

administrator - controls trading to restrict service usage to importers acting on behalf of authorized individuals

D.3 COMPATIBILITY OF INTERFACE SPECIFICATIONS

The definition of types as sets yields a simple definition of subtypes - i.e., A IsASubtypeOf B is equivalent to A is a subset of B. Since set membership is defined in terms of satisfying the specification of the type, then an instance of A (satisfying the specification of A) also is an instance of B (satisfying the specification of B). The interface specification for A is compatible with the interface specification for B.

Interface specifications describe required behaviour; if specification A is compatible with specification B, then A instances behave as required by specification B; in addition, they also behave as required by specification A.

For an association to be established, the interface specification of the exporter in an export must be compatible with the interface specification of the importer in an import.

D.4 EPOCHS

There are several different times in the life of a distributed system at which trading can occur. The only requirement is that an association must be established prior to the first operation invocation. These epochs are:

(a) construction: when objects are assembled - e.g. the associations between objects can be fixed by a compiler or link editor;

(b) configuration: when objects are bound to resources - e.g. if a configuration editor is used to distribute portions of an application, it can fix associations between objects;

(c) initialization: when objects are instantiated - e.g. the associations between objects can be fixed by a program loader based upon contextual information provided to the loader;

(d) execution: when objects are active - e.g. imports and exports result in requests to a third party trading system.

D.5 INTERFACE TRADING

This subclause describes the primitives used for initiating interface trading and the criteria by which a match between imports and exports is established.
There are two primitives used in establishing an association between objects: export and import. To announce its availability, an object exports one of its interfaces. The export is registered in the trading system and assigned a permanently unique identifier. Subsequently, another object (it can be the same object) can issue an import. If there is an exported interface specification compatible with the imported interface specification, then an association can be established between the objects concerned.

Figure 34 shows the entities involved in establishing an association between two objects, A and B between which there is a client-server relationship. Let us assume that B is the exporter and A is the importer. Each object has a requester which issues the export or import (it may be part of the object, or a separate object - in the case of non-execution epoch trading, the requester is a different object). In addition, an object has a specification of the interface(s) it supports/uses. The exporter is concerned with the responsibilities of the performer role in the interface specification, while the importer is concerned with the responsibilities of the requester role in the interface specification.

B's requester initiates an export by sending B's interface specification to the trading service. Subsequently, A's requester can request to import the service. The success of an import is determined by the policy associated with the trading service. This policy is set by the trading service authority. Essentially, it consists of the compatibility of interface specifications and the contexts available to requesters (see subclause D.9).

The result of a successful trade causes an association to be established between the objects. This is achieved by the trading service setting up the interface adaptors of the objects. Subclause D.9.2 discusses some of the necessary mechanisms.

The objects can also be object groups. The principle is the same, except that exports and imports are issued on behalf of a group of objects and not individual objects. Also, the interface adaptors are more complex when providing access to the object groups. The trading service needs to be aware of the group structure and will interact with the group manager.
D.6 TYPES AND PROPERTIES

An association can only be established between compatible interface specifications. As already mentioned, the interface specification for A is compatible with the interface specification for B if A is a subtype of B. Thus, an instance of interface A can be substituted for an instance of interface B. This allows the structuring of the interface type space. However, to fully define all the possible interface types is likely to result in an unacceptable expansion of the type spaces. For example, consider the typing of printer interfaces:

- Paper size: A4, Continuous, etc.
- Speed: Pages per minute, characters per second, etc.
- Command Language: Postscript\textsuperscript{TM}, Interpress\textsuperscript{TM}, etc.
- Location: Various levels of granularity.
- Cost: Price per page, price per character, etc.

Defining subtypes of the PRINTER interface to cover all the combinations of attributes would result in a very large number of interface types. Instead, the notion of property is introduced. The specification defines the properties an instance must have, but not their values. These are defined by the instances themselves. For example, the interface specification PRINTER might state that all printers have an attribute which is the paper size property. Individual instances of the PRINTER interface will assign a value to this property.

The division between a type and property is arbitrary. Both the type and property values form the contract of the service offer. However, property values are more flexible. They can be changed between trading operations. For example, a PRINTER may change the paper size property between jobs.
Since properties of an *instance* are defined in the *interface specification*, all *instances* of a compatible *interface* must also assign values to the properties defined in the supertype. For example, the primitive type *OBJECT* might have the properties 'location' and 'owner'. If all other *types* in the systems are subtypes of *OBJECT*, then they all must have the properties "location" and "owner". Note that the value associated with a property is defined by each *instance*.

Property values have units. For example, *paper size* is an enumerated *type* containing A4, Continuous, etc. while *speed* is *pages-per-sec*. The appropriate units are specified with the property in an *interface specification*.

As properties are part of the *service* contract in an exported *interface*, it is necessary to ensure that they have the correct values. For example, the value of 'location' or 'owner' of an *object* should not be forgeable. Therefore, we need some mechanism to ensure the correct values are defined. For 'location' and 'owner', this can be provided by the *object* support environment. However, other properties may have their values set by other trusted agents in the system.

It may be necessary to distinguish between trustworthy and untrustworthy properties. The values of the former properties are guaranteed to be correct and therefore need to be set by trusted mechanisms. However, the latter properties can be set by the *objects* themselves.

**D.7 TRADING CONTEXTS AND TRADING SCOPES**

Configuration management policy defines the *instances* between which associations can be established. As *trading* is part of configuration management, it must enforce configuration policy. The effect is that an *import* can not match just any compatible *interface*. Instead, the *export* space is divided into *trading contexts* (abbreviated *contexts*).

*Contexts* may be thought of as directories. They contain references to exported *services*. Also, they can reference other *contexts*. Therefore, we define a *trading scope* as the set of *contexts* reachable from a given *context*, including the initial *context*.

The *export* contains a parameter that defines the *context* in which the exported *service* should be registered. Similarly, the *import* contains a reference to a *context*. Unless a matching *export* has been registered in a *context* included in the *scope* specified in the *import*, the *import* will fail.

The structure of *contexts* is arbitrary, being determined by *administrators* of the *trading system*. However, there are advantages in limiting the structure to hierarchies or directed acyclic graphs; e.g. it simplifies *searching* *trading scopes* as there are no cycles, and it is easier to determine that the *context space* is divided into separate *scopes*.

**D.8 CONSTRAINTS MATCHING**

The aim of *interface trading* is to select the most suitable *export* for a particular *import*. There may be several *exports* of compatible *type* in the *import* requester's *scope*. This set of *exports* can be further subdivided by specifying
constraints - e.g. the nearest printer that accepts Postscript™ and costs less than
5p per page.

NOTE D.1:
A language for expressing trading constraints is outlined here. For ODP purposes it may be
appropriate to use an existing query language; e.g. SQL.

The importer supplies a constraints expression that is used to select the most
suitable offer. The constraints expression consists of:

- Superlative functions: min, max.
- Comparative functions: Greater-than (＞), Less-than (＜), equal-to (＝), etc.
- Constructors: "And" (＆), "Or" (｜), "Followed-by" (→).

Both superlative and comparative functions are applied to property values, the
difference being that comparative functions are applied to individual service
exports in the importer's scope, while superlative functions are applied to a set of
exports. If there are two of more service exports with the same property values in
a min or max function, then the trading system chooses one at random.

Constructors allow several constraint functions to be applied to the service
exports. "And" (＆) and "or" (｜) have their usual logical meanings. "Followed-
by" (→) is used to give precedence to the constraints expression.

These constructs allow flexible constraints expressions to be defined. For
example, consider selecting the cheapest A4 printer in the ECMA offices that
accepts Postscript™ commands.

((location = MRC) & (command = POST) & (size = A4)) → min(cost)

This constraint expression is applied to all service exports registered in the
importer's scope that are of type PRINTER. The first three parts are comparative
functions that determine that the printer is at MRC, accepts Postscript™
commands, and uses A4 paper. The result is a set (possibly null) of service
exports which meet these constraints. The superlative function min is then
applied to the cost of each printer in the set.

Both comparative and superlative functions can be applied to mathematical
functions of property values. For example, consider the choice between the
fastest and cheapest printers. A weighted expression can be used as shown
below:

min(W1 * cost + W2 * speed)

This expression will select the minimum of cost weighted by W1 and speed
weighted be W2. The mathematical operators +, -, *, and / are permitted in
constraint expressions. Use of trigonometric and logarithmic functions are also
permitted.

As stated earlier, property values have units. To be used in constraint
expressions, the comparative and superlative functions must be defined on all
possible units. For example, it must be possible to determine if the value of a
property in one export is greater than that same property in another export. This requires overloading of the constraint functions.

In addition, it must be possible to convert property units to some base unit for multiple property comparisons. Thus the minimum of cost and speed for the PRINTER interface could be specified as:

$$\text{min}(W1 \times \text{ord(cost)} + W2 \times \text{ord(speed)})$$

where ord is a function defined on all numeric-based properties that converts values to real numbers.

**D.9 MECHANISMS**

**D.9.1 Support environment primitives**

The three main primitives provided by the support environment for interaction with the trading system are:

- **EXPORT**
  
  $$\text{exportHandle} \leq \text{export(InterfaceSpecification, TradingContext, PropertyValues)}$$

- **IMPORT**
  
  $$\text{importHandle} \leq \text{import(InterfaceSpecification, TradingContext, Constraints)}$$

- **SEARCH**
  
  $$\text{setOfExport} \leq \text{search(InterfaceSpecification, TradingContext, Constraints)}$$

where:

- **InterfaceSpecification** is the interface specification upon which any resulting association(s) will be based.

- **TradingContext** is the naming context in which the export is to be placed, or the anchor of the trading scope in which the search/import is to take place.

- **PropertyValues** provides the values for the properties associated with the service export.

**Constraints** specifies the matching constraints to be applied in the search/import

An association results from an import if the following conditions are met:

- \text{InterfaceSpecification}^{\text{export}} \text{IsCompatibleWith} \text{InterfaceSpecification}^{\text{import}}

- TradingContext^{export} is in the Trading Scope of TradingContext^{import}

- at least one export matches the Constraints.

The search primitive returns the unique identifiers of all exports which satisfy the above criteria. The importer can then associate itself with a particular export by an import with InterfaceSpecification and TradingContext unchanged, but with a Constraints expression of the form

"UniqueId = UI"
where UI is the unique identifier for the chosen \textit{export}. Note that this method is only of interest in the execution epoch.

D.9.2 Epochs, associations, and interface adaptors

The structural model shown in Figure 35 may aid visualization of the interplay between \textit{epochs}, \textit{associations}, and \textit{interface adaptors}.

The vector labeled \textit{ih} represents an \textit{association} to another \textit{object} providing a particular \textit{service}. The additional vectors represent other engineering choices which must be made to realize the \textit{association}. These additional vectors are details of the \textit{interface adaptor} functionality described in clause 23.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{dispatch-vector.png}
\caption{Dispatch vector}
\end{figure}

Each of these vectors may be filled in at different \textit{epochs}. Some are fixed at system design time (e.g. standard intermediate over-the-wire (OTW) format will always be used), construction time (e.g. service is local, so dispense with remote invocation machinery), instantiation time (e.g. protocol vectors are determined from the network environment known to the support environment), and execution time (e.g. OTW format negotiated at bind time).
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract interface</td>
<td>13.2</td>
</tr>
<tr>
<td>access transparency</td>
<td>B.4.2</td>
</tr>
<tr>
<td>ACID properties</td>
<td>23.2</td>
</tr>
<tr>
<td>action</td>
<td>B.6</td>
</tr>
<tr>
<td>activity</td>
<td>B.6</td>
</tr>
<tr>
<td>address</td>
<td>22.2</td>
</tr>
<tr>
<td>administrator</td>
<td>D.2</td>
</tr>
<tr>
<td>application component</td>
<td>8.2</td>
</tr>
<tr>
<td>application program</td>
<td>B.2.2</td>
</tr>
<tr>
<td>Aspect</td>
<td>B.5</td>
</tr>
<tr>
<td>association</td>
<td>D.2</td>
</tr>
<tr>
<td>atomic object</td>
<td>23.2</td>
</tr>
<tr>
<td>basic distribution transparencies</td>
<td>23.2</td>
</tr>
<tr>
<td>binding</td>
<td>22.2</td>
</tr>
<tr>
<td>capsule</td>
<td>19.2</td>
</tr>
<tr>
<td>class</td>
<td>D.2</td>
</tr>
<tr>
<td>client</td>
<td>21.2</td>
</tr>
<tr>
<td>client-server interaction</td>
<td>21.2</td>
</tr>
<tr>
<td>client-server model</td>
<td>21.2</td>
</tr>
<tr>
<td>component</td>
<td>B.2.2</td>
</tr>
<tr>
<td>Computation viewpoint</td>
<td>B.5</td>
</tr>
<tr>
<td>computer application</td>
<td>8.2</td>
</tr>
<tr>
<td>concurrency transparency</td>
<td>B.4.2</td>
</tr>
<tr>
<td>connexion</td>
<td>B.6</td>
</tr>
<tr>
<td>consumer</td>
<td>21.2</td>
</tr>
<tr>
<td>context relative name</td>
<td>22.2</td>
</tr>
<tr>
<td>distributed application</td>
<td>8.2</td>
</tr>
<tr>
<td>distributed information system</td>
<td>B.2.2</td>
</tr>
<tr>
<td>distributed IT system</td>
<td>B.2.2</td>
</tr>
<tr>
<td>distributed operating system</td>
<td>C.2</td>
</tr>
<tr>
<td>distributed processing</td>
<td>5. B.1.2</td>
</tr>
<tr>
<td>distributed system</td>
<td>B.2.2</td>
</tr>
<tr>
<td>distribution transparency</td>
<td>B.4.2</td>
</tr>
<tr>
<td>distribution transparency utilities</td>
<td>24.2</td>
</tr>
<tr>
<td>DT0; DT1; DT2; DT3:</td>
<td>24.2</td>
</tr>
<tr>
<td>early binding; static binding</td>
<td>22.2</td>
</tr>
<tr>
<td>Engineering viewpoint</td>
<td>B.5</td>
</tr>
</tbody>
</table>
Enterprise viewpoint  B.5
export  21.2  D.2
exporter  21.2  D.2
extended distribution transparencies  23.2
failure transparency  B.4.2
federated trading systems  D.2
final form model  13.2
import  21.2  D.2
importer  21.2  D.2
information system  B.2.2
information technology system  B.2.2
Information viewpoint  B.5
instance  D.2
inter-process communication (IPC)  20.2
interaction  B.6
interface adaptor  23.2
interface object  13.2
interface specification  D.2
interpreter  19.2
IT system  B.2.2
joint action  B.6
late binding; dynamic binding  22.2
location transparency  B.4.2
management utilities  24.2
migration transparency  B.4.2
model  B.6
name  22.2
naming context  22.2
naming tree  22.2
native component  13.2
native interpreter  19.2
native name  22.2
natural system  B.2.2
network operating system  C.2
networking  C.2
networking facilities  C.2
nucleus  24.2
object  B.6
object diagram  B.6
object factory  23.2
object factory utilities  24.2
object group  23.2
object model  B.6
ODP association  22.2
ODP component  13.2
ODP interface  13.2
ODP name  22.2
open distributed processing (ODP)  5. B.1.2
operating system  C.2
processing  5. 19.2 B.1.2
processing component  13.2
producer  21.2
producer-consumer interaction  21.2
producer-consumer model  21.2
Reference Model of Open Distributed Processing (RM-ODP)  B.1.2
replication transparency  B.4.2
SE-ODP interpreter  19.2
SE-ODP logical unit  24.2
SE-ODP processing model  19.2
SE-ODP runtime  24.2
SE-ODP utility  24.2
security utilities  24.2
server  21.2
service  21.2
service interaction  21.2
social system  B.2.2
socio-technical system  B.2.2
specification  D.2
subsystem  B.2.2
Support Environment for Open Distributed Processing (SE-ODP)  5.
system  B.2.2

technical system  B.2.2
Technology viewpoint  B.5
trader  21.2 D.2
trading  21.2 D.2
trading context  D.2
trading domain  D.2
trading scope  D.2
trading service  21.2 D.2
trading system  D.2
trading utility  24.2
type  D.2
viewpoint  B.5