The ISO Reference Model for Open Distributed Processing - An Introduction

Kazi Farooqui[1], Luigi Logrippo[1], Jan de Meer[2],

1)Department of Computer Science, University of Ottawa,
Ottawa K1N 6N5, Canada
email: (farooqui | luigi)@csi.uottawa.ca

2)Research Institute for Open Communication Systems Berlin
(GMD-FOKUS), D10623 Berlin, Hardenbergplatz 2, Germany,
email: jdm@fokus.berlin.gmd.d400.de

Abstract:

The ISO Reference Model of Open Distributed Processing (RM-ODP) consists of four parts - an Overview of the reference model, the Descriptive Model, the Prescriptive Model, and the Architectural Semantics. The four parts provide the concepts and rules of *distributed processing* to ensure *openness* between interacting distributed application components. Openness is a combination of characteristics, i.e. scalability, accessibility, heterogeneity, autonomy and distribution.

The RM-ODP introduces the concept of *viewpoint* to describe a system from a particular set of concerns, and hence to deal with the complexity of distributed systems. While all the viewpoints are relevant to the description and design of distributed systems, the computation and engineering models are the ones that bear most directly on the design and implementation of distributed systems. From a distributed software engineering point of view, the computational and engineering viewpoints are the most important; they reflect the software structure of the distributed application most closely. In this introductory paper, we concentrate on the computational and engineering viewpoints.

Keywords:
Standardisation, Distributed Processing, Viewpoint Models,
Architectural Semantics, Specification Process, Openness

1. INTRODUCTION

The standardization effort, within ISO and ITU-T (formerly CCITT) on the Reference Model for Open Distributed Processing (RM-ODP) started in response to the diverse forces affecting the development and evolution of distributed computing systems.

The ISO and ITU-T have actively investigated the problem of heterogeneous system *interconnection.* This has resulted in the Open System Interconnection (OSI) model. This model, structured into seven layers, is widely used for heterogeneous system interconnection. With the proliferation of numerous application-specific standards in the layer 7, the application layer, of the OSI model, it was soon realized that the (lower six layers of the) OSI stack only provides support for communication, and that a more general reference model is needed to address all aspects related to distribution of applications.

The need to address application *intercommunication* problems rather than pure system *interconnection* problems, in a standardized way, was very strong in the distributed systems community. In the last few years, the needs for openness, integration, and inter-operability of distributed applications were strongly felt in order to foster the widespread use of distributed processing applications. This, together with the demands of the open service market (based on networked computing) for standardized interfaces between application components and the supporting distributed platform, paved the way towards the standardization of an Open Distributed Processing Reference Model (RM-ODP) as the basis for the provision of these requirements. The RM-ODP addresses the need of models for the development of complex distributed systems. The idea of ODP is an attempt to provide a standardized framework for the establishment of an integrated distributed environments that spans heterogeneous systems. In contrast to OSI, ODP is not restricted to communication between heterogeneous systems. It deals also with application portability across systems and with the provision of various distribution transparencies within systems. In this sense, ODP encompasses, and extends OSI. OSI becomes a (*communication)* enabling technology for ODP applications.

The ODP reference model is quite general and can be used in several areas. It is a generic framework for the development of numerous future standards in various application domains. Specific fields of ODP applications include advanced telecommunications, Intelligent Networks, Automated Manufacturing Systems, Office Systems, Management Information Systems, etc.

The model identifies several types of interfaces at which standardization may be required, and places constraints only at these interfaces. It identifies the functionality of the distributed platform, the ODP Support Environment, required for open and distribution-transparent interaction between application components.

The ISO Project JTC 1.21.49, i.e. Basic Reference Model of Open Distributed Processing (RM-ODP) is a long range, ongoing, joint standardisation activity of ISO and ITU-T. It is expected that the International Standard (ISO 10746) of the RM-ODP will be available by the end of 1996.

2. PARTS OF ODP REFERENCE MODEL

The set of documents, which build the forthcoming standard of the RM-ODP consist of four parts. Part 1 of the RM-ODP (ISO 10746-1/ ITU-T X.901) provides an overview and a guide to the use of other parts. It introduces the concept of information distribution. The application of standards for distributed information processing ranges from global communication networks to applications that run on them. It includes all types of application intercommunication and information media such as data, text, voice, video, hyper-media, etc. and all types of communication facilities. Thus, standardization of distribution aspects, i.e. processing, storage, user access, communication, interworking, identification, management and security aspects, supports the portability of applications and the interworking between ODP systems.

Part 2 (ISO 10746-2 / ITU-T X.902) and Part-3 (ISO 10746-3 / ITU-T X.903) respectively, are the descriptive and prescriptive models of the RM-ODP. Part 2 provides the basic modelling concepts, whereas part 3 prescribes the concepts, rules and functions a system must adhere to in order to be qualified as an ODP System. The concepts, rules and functions are structured according to the RM-ODP concept of the *"viewpoint"*.

For each of these viewpoints, viewpoint-specific *"languages"* are introduced in Part-3, that use the terminology (descriptive concepts) of the ODP Part 2 in order to define viewpoint-specific concepts and rules. Apart from the viewpoint languages, the ODP functions such as distribution transparency functions, security functions, management functions, etc. are defined in Part-3. These functions constitute the building blocks of ODP systems (and are subject to separate standardization). The viewpoint approach is applied for the specification to the ODP functions.

The object concept plays an important role in the modelling of ODP systems. An object-oriented approach has been adopted for modelling distributed systems in each viewpoint. An object stands for data abstraction, function encapsulation and modularity. However, different interpretations of the concept of an object are possible, i.e. a real-world thing, the subject of concern, an idealised thing, a denotation of a model or program or the object itself as part of the real-world.

Part-4 (ISO 10746-4 / ITU-T X.904), of the RM-ODP is called "Architectural Semantics". It deals with how the modelling concepts of Part-2 and the viewpoint languages of Part-3 can be represented in standardised formal description techniques (FDTs) such as LOTOS, Estelle, and SDL. There are, however, also suggestions of including non-standardized specification languages like Z and RAISE RSL. It appears that no single specification language is suitable for specifying systems in all viewpoints.

3. THE VIEWPOINT APPROACH OF RM-ODP

For any given information processing system, there are a number of user categories - or more accurately, a number of *"roles*" - that have an interest in the system. Examples include the members of the enterprise who use the system, the system analysts, who specify it, the system designers, who implement it, and the system administrator, who install it. Each role is interested in the same system, but their relative views of the system are different, they see different issues, they have different requirements, and they use different vocabularies (or languages) when describing the system. RM-ODP attempts to recognize these different interests by defining different *viewpoints*.

Rather than attempting to deal with the full complexity of distributed systems, the RM-ODP considers the system from different viewpoints or projections, each of which is chosen to reflect one set of design concerns. Each viewpoint represents a different abstraction of the original distributed system, without the need to create one large model describing it.

The ODP framework of viewpoints partitions the concerns to be addressed in the design of distributed systems. A viewpoint leads to a representation of the system with emphasis on a specific set of concerns, and the resulting representation is an abstraction of the system, that is, a description which recognizes some distinctions (those relevant to the concern) and ignores others (those not relevant to the concern). Different viewpoints address different concerns, but there is a common ground between them. The framework of viewpoints must treat this common ground consistently, in order to relate viewpoint models and to make it possible to assert correspondences between the representations of the same system in different viewpoints. This framework allows the verification of both the completeness of the various descriptions and of the consistency between them.

The ODP viewpoints can be used to structure the specification of a distributed system, and can be related to a design methodology. Design of the system can be regarded as a process that may be subdivided into phases related to different viewpoints. Each of the viewpoints can be used as problem analysis technique as well as a solution space of the relevant issues of the problem domain.

These viewpoints should not be seen as architectural layers, but rather as different abstractions of the same system, and should all be used to completely analyse the system. With this approach, consistent and complete system models may be described and developed based on concepts and methods still to be designed for individual viewpoints.

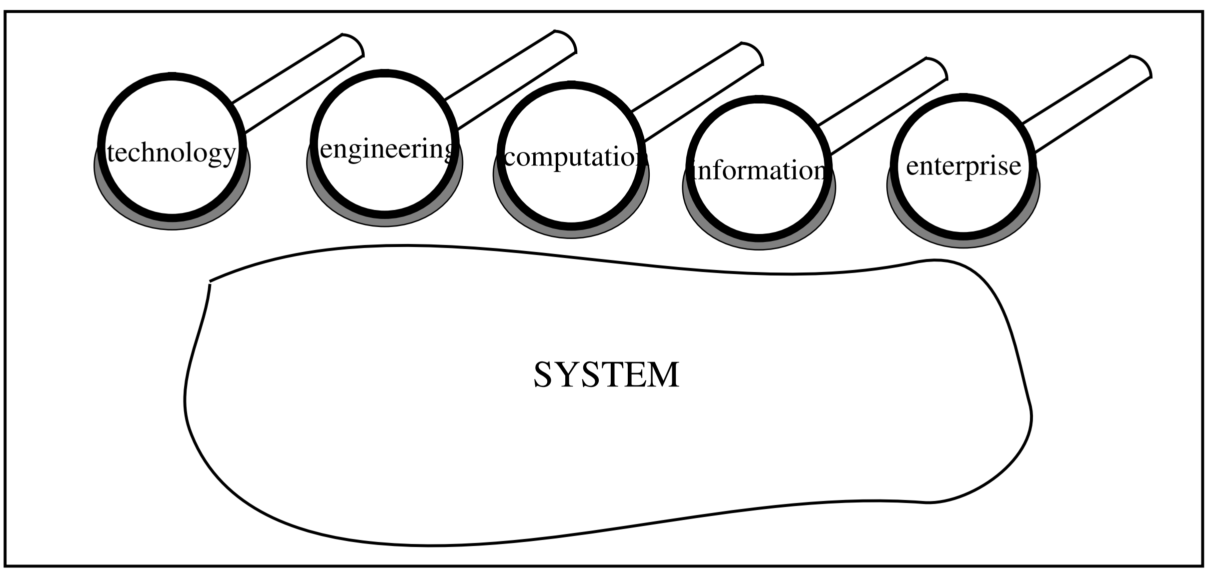

Figure 1. Viewpoints: Different Projections on the System

RM-ODP defines the following five viewpoints. Together they provide the complete description of the system: *enterprise viewpoint, information viewpoint, computational viewpoint, engineering viewpoint*, and *technology viewpoint*. The concerns addressed in each of the viewpoints are briefly sketched below:

1. Enterprise Viewpoint: It is directed to the *needs* of the *users* of an information system. It describes the (distributed) system in terms of answering what it is required to do for the enterprise or business. It is the most abstract of the ODP framework of viewpoints stating high level enterprise requirements and policies.

2. Information Viewpoint: It focuses on the information content of the enterprise. The information modelling activity involves identifying *information elements* of the system, *manipulations* that may be performed on information elements, and the *information flows* in the system.

3. Computational Viewpoint: It deals with the logical partitioning of the distributed applications independent of any specific distributed environment on which they run. It *hides* from the application designer the details of the underlying machine (distributed platform) that supports the application.

4. Engineering Viewpoint: It addresses the issues of system support (platform) for distributed applications. It identifies the functionality of the distributed platform required for the support of the computational model.

5. Technology Viewpoint: The technology model identifies possible technical artifacts for the engineering mechanisms, computational structures, information structures, and enterprise structures.

The purpose of viewpoints of the RM-ODP is to position services relative to one another, to guide the selection of appropriate models of services, and to help in the placement of boundaries upon ODP. The framework of viewpoints is used to partition the concerns to be addressed when describing all facets of an ODP system, so that the task is made simpler. A summary of ODP viewpoints is presented in table 1.

Table 1: Summary of ODP Viewpoints

Viewpoint	Enterprise	Information	Computation	Engineering	Technology
Areas of concern.	Enterprise needs of IS; Objectives and roles of IS in the organization.	Information models, Information structures, Information flows, Information manipulation.	Logical partitioning of application, application components, component interfaces, component interactions; service-oriented view of distributed application.	Distributed platform infrastructure;distribution transparency, communication support, and other distribution enabling, regulating, and hiding generic mechanisms; system-oriented view of distributed application.	Technological artifacts required for realizing engineering mechanisms.
Main concepts	agents, artifacts, communities, roles, etc.	schemas, relations, integrity roles, etc.	computational object, computational interface, environment constraints, computational interactions, etc.	Basic engineering objects, transparency objects, protocol object, nucleus, etc.	Technological solutions corresponding to engineering mechanisms and structures

Table 1: Summary of ODP Viewpoints

Viewpoint	Enterprise	Information	Computation	Engineering	Technology
Whom does it concern	System procurers, Corporate managers.	Information Analysts, System Analysts, Information Engineers.	Application designers and programmers.	Operating System designers, Communication System designers, System designers.	System integrators, System vendors.
Language/ Notation	requirement description languages.	entity-relationship models, conceptual schemas, etc.	application programming environments, tools, programming languages, etc.	Distributed platforms, engineering support environments, etc.	Technology mappings, identification of technical artifacts, etc.
Role in software engineering	Requirement capture and early design of distributed system.	Conceptual design and information modelling.	Software design and development.	System design and development.	Technology identification, procurement, installation.

Using the five ODP viewpoints to examine system issues encourages a clear separation of concerns, which in turn leads to a better understanding of the problems being addressed: describing the role of the enterprise (enterprise viewpoint) independently of the way in which that role is automated; describing the information content of the system (information viewpoint) independently of the way in which the information is stored or manipulated; describing the application programming environment (computation viewpoint) independently of the way in which that environment is supported; describing the components, mechanisms used to build systems independently of the machines on which they run; and describing the basic system hardware and software (technology viewpoint) independently of the role it plays in the enterprise.

4. THE ODP COMPUTATIONAL MODEL

The ODP computational model is a framework for describing the structure, specification and interactions of (components of) a distributed application on a (distributed) computing platform, which is also called the computational infrastructure.

The computational model provides a set of basic (abstract) concepts and elements for the construction of a programming (specification) language for which the model does not provide any syntax. Using the computational modelling concepts, one can specify (program) a distributed application without worrying about the details of the underlying distributed execution platform. The design principle of the computational model is to minimize the amount of engineering details that the application programmer is required to know, yet at the same time allowing the programmer to exploit the benefits of distributed computing.

A *computational specification* of a distributed application consists of the composition of *computational objects* (which represent application components) interacting, by *operation invocations*, at their interfaces. It identifies the activities that occur within the computational objects, and the interactions that occur at their interfaces, (*computational interfaces*).

4.1 Computational Model: A Object-Oriented View of Distributed Application

The computational model is based on a distributed-object model. It prescribes an object-oriented view of the distributed application. Applications are collections of interacting objects. In this model, objects are the units of distribution, encapsulation, and failure.

The computational model is an 'object world' populated with concurrent (computational) objects interacting with each other, in a *distribution-transparent* abstraction, by invoking operations at their interfaces. An object can have multiple interfaces and these interfaces define the interactions that are possible with the object.

"Activity" is a unit of concurrency within an object. A collection of (computational) objects may have any number of activities threading through them. The state encapsulated by the object can be accessed and modified by the activities executing the operations in the interfaces of that object.

A distributed computation progresses by operation invocations at object interfaces. The activity in an object (invoking object) can pass into another object (invoked object) by invoking operations in the interface of the invoked object. Activities carry the state of their computations with them, i.e., when an activity passes into an operation it carries the parameters for that invocation, and returns carrying results. In the computational model, concurrency within an object and communication between objects are separate concerns. While concurrency is modelled by the concept of activity, communication between object is modelled as (remote) invocation of an operation.

4.2 Distribution Issues and the Computational Model

Computational specifications are intended to be distribution-transparent, i.e., written without regard to the specifics of a physically distributed, heterogeneous environment. However, the expression of *environment constraints* in the computational interface template provides a hint of the application requirements from the distributed platform, e.g., distribution transparencies, security mechanisms, specific resource requirements, etc.

At the computational level, user applications are unaware of how the underlying distributed platform is structured or how the distribution enabling and regulating mechanisms are realised.

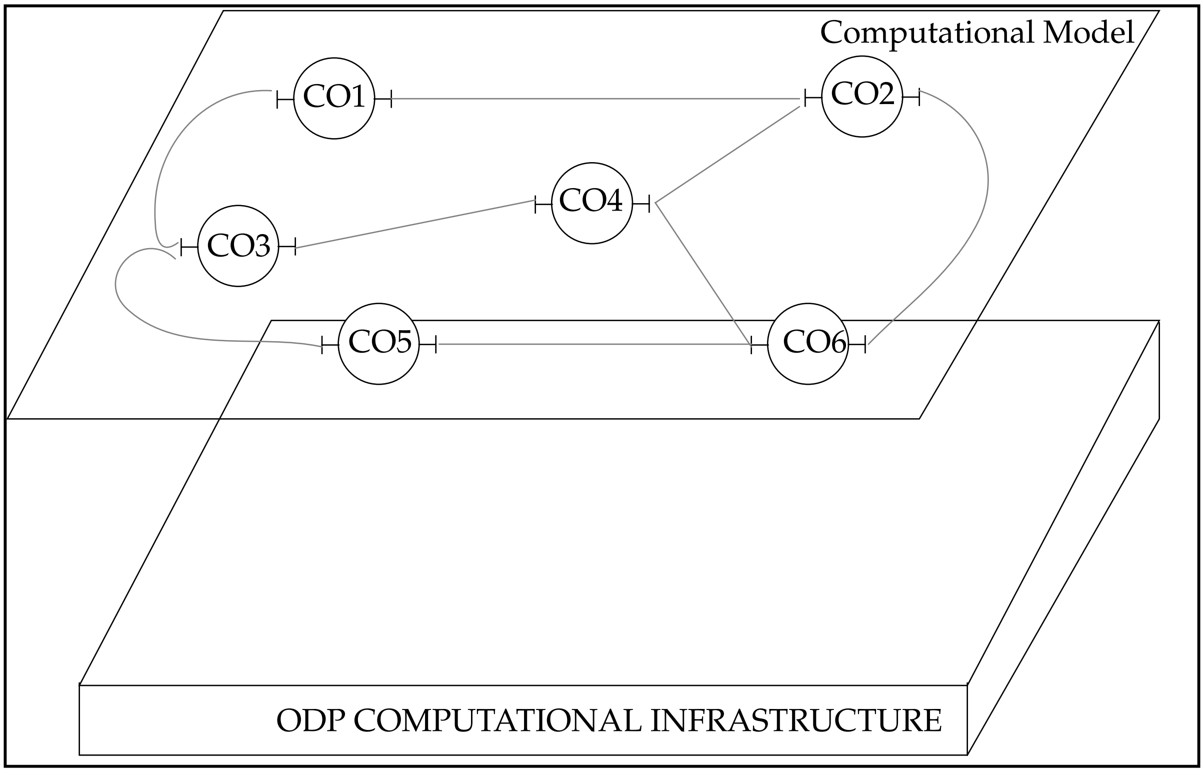

Figure 2. ODP Computational: An Object World supported by Distributed Platform

4.3 Elements of the Computational Model

The design philosophy of the computational model has been to find the smallest number of concepts (elements) needed to describe distributed computations and to propose a *declarative* approach to the formulation of each concept. This section is a brief introduction of some basic computational elements out of which the *computational specification* of the distributed application is constructed.

The basic elements of the computational model are: *computational object*, *computational interface*, interface invocation mechanisms such as *computational operation*, and the abstraction to model the communication between the computational interfaces- *binding object*.

Computational Object: The components of distributed application are represented as computational objects in the computational model. The computational objects are the units of (application) structure and distribution. The computational objects model both the application components that perform information processing and those components that store the information.

As shown in figure 3, a computational object template consists of a set of computational interface templates which the object can instantiate.

Computational Interface: While computational objects are the units of structure and encapsulation of (application-specific) services, interfaces are the units of provision of services; they are the places at which objects can interact and obtain services.

The distributed application components (modelled as computational objects) may be written in different (programming) languages and may run on heterogeneous environments. In order for a component to be constructed independently of another component with which it is to interact, a precise specification of the interactions between them is necessary. The specification of interaction between computational objects and their environment, and of their requirements of interaction are captured by interfaces. The computational interfaces model different interaction concerns of an object.

A computational object may support multiple computational interfaces which need not be of the same type. Interfaces of the same type may be provided by objects which are not of the same type. Each object may provide interfaces which are unlike those provided by the other object.

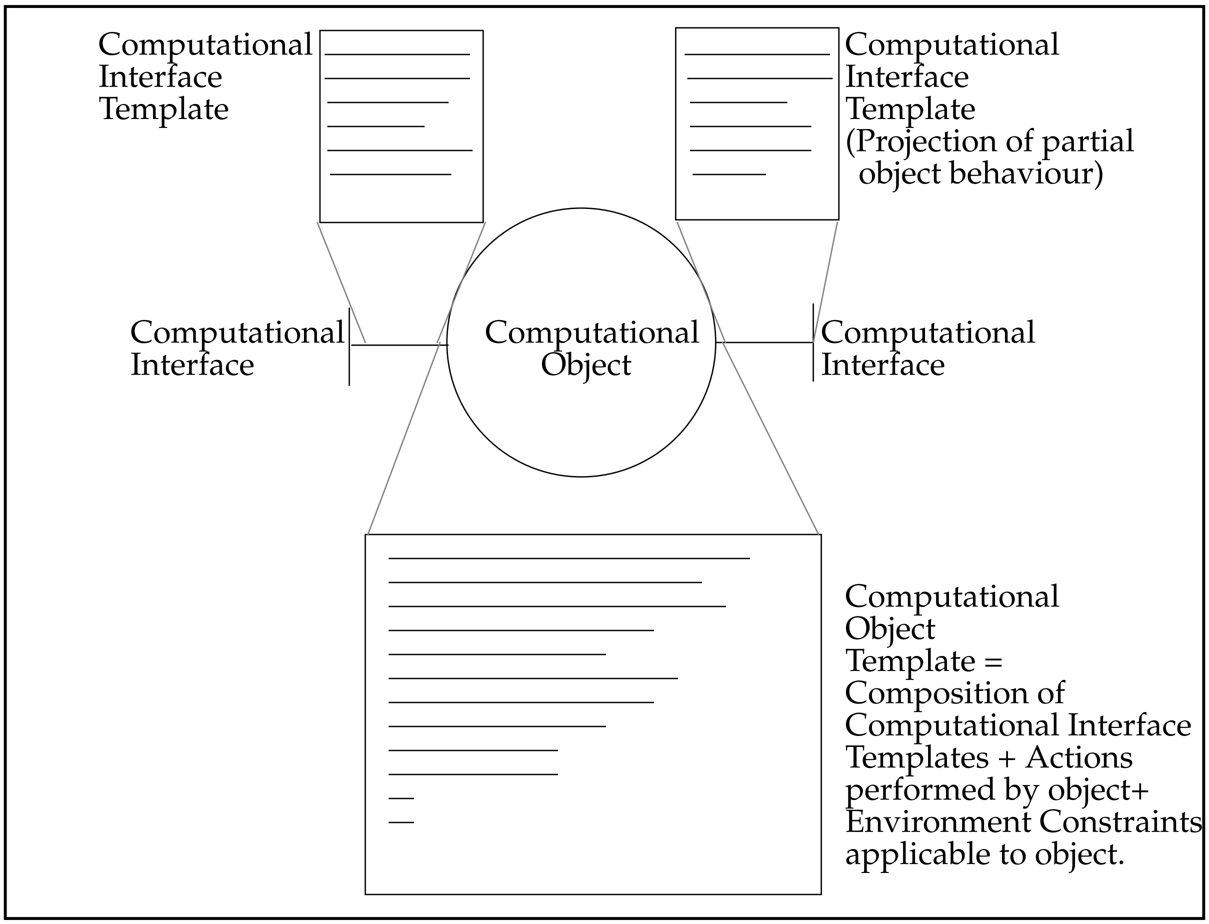

Figure 3. ODP Computational Model Concepts

In the ODP computational model two kinds of interfaces are identified: **operational interfaces** and **stream interfaces**.

Operational Interface: The specification of operational interface template consists of:

1. Operation Specification
2. Behaviour Specification
3. Environment Contract

The **operation specification** includes the operation name together with the number, sequence, and type of arguments that may be passed in each operation invocation and its response(s). This is called *operation signature*.

The **behaviour specification** defines the behaviour exhibited at the interface. All possible orderings of operation invocations at or from the interface are specified. The behaviour constitutes the protocol part of the interface.

Most interface specifications, to date, have concentrated on the syntactic requirements of the interface such as the operation signature. Aspects other than pure syntax are also important in facilitating the interaction between a pair of objects. This additional semantic information falls into two categories:

* information affecting the way in which the infrastructure supports the interactions; this information constrains the type of distribution transparencies, choice of communication protocols, etc. that must be placed in the interaction path between the interacting objects.

* the behaviour (or the semantics) of the service offered at the interface; an interface is viewed as a projection of an object's behaviour, seen only in terms of a specified set of observable actions. As a result, signature compatibility is less discriminating than interface compatibility.

The environment contract in the computational interface template defines the following attributes:

1. distribution transparency requirement on operation invocation.
2. quality of service (including communication quality of service) attributes associated with the operations.
3. temporal constraints on operations (e.g., deadlines).
4. dependability constraints (e.g., availability, reliability, fault tolerance, security etc.)
5. location constraints on interfaces (and hence their supporting objects).
6. other environment constraints on operations (e.g., those arising from enterprise and information viewpoint).

These attributes may be associated with individual operations or the entire interface. The environment contract is an important component of the computational interface template and has a direct relationship to the realized engineering structures and mechanisms.

Stream Interface: The computational objects may perform both the information processing task as well as act as containers of information. There is a need to model not only the interfaces which provide 'service', but also those interfaces which model 'continuous' information flow. Such interfaces are modelled, in the computational model, as *stream interfaces* (also known as *non-operational interfaces).*

The stream interface is a set of information flows whose behaviour is described by a single action which continues throughout the life time of the interface. Information media such as voice and video inherently consists of a continuous sequence of symbols. Such media are described as *continuous* and the term *stream* is used to refer to the sequence of symbols comprising such a medium.

Examples include the flow of audio or video information in a multimedia application, or the continuous flow of periodic sensor readings in a process control application. The computational description does not need to be concerned with detailed mechanisms; the fact that the flow is established and continues during the relevant period is enough.

The template for a non-operational or stream interface consists of:
Stream Signature: A specification of the type of each information flow contained in a stream interface and, for each flow, the direction in which the flow takes place.

Environment Constraint: Continuous media have strict timing and synchronization requirements. The environment constraints that are relevant to stream interfaces include synchronization and clocking properties, time constraints, priority constraints, throughput, jitter, delay, media-specific communication quality requirements, etc., in addition to the properties applicable to operational interfaces.
Role: A role for each information flow, e.g., a producer object or a consumer object.

Binding Object: Interactions between computational objects are only possible, when their interfaces are bound. There is a concept of implicit and explicit binding in the computational model. When objects get implicitly bound in the computational model, it is assumed that the underlying platform (the engineering infrastructure), will provide the service of checking the consistence between the interfaces to be bound.

The computational objects are explicitly bound through a binding object. The template for the binding object specifies the interaction patterns between the bound computational objects. The binding object contains control interfaces which allow dynamic modification of number and types of objects involved in the binding.

5. ENGINEERING MODEL

The engineering model is an abstract model to express the concepts of the engineering viewpoint. It involves concepts such as operating systems, distribution transparency mechanisms, communication systems (protocols, networks), processors, storage, etc. As the notions of processor, memory, transport network play a more indirect role in a distributed system, the term ‘engineering model’ is used here in a more general way to describe a framework oriented towards the organization of the underlying distributed infrastructure and targeted to the application support. It mostly focuses on what services may be provided to applications and what mechanisms should be used to obtain these services. The term *platform* is used to refer to the (configuration of) services offered to applications by the infrastructure.

The engineering model is still an abstraction of the distributed system, but it is a different abstraction than the computational model. Distribution is no longer transparent, but we still need not concern ourselves with real computers or with the implementations (technology) of mechanisms or services identified in the engineering model. The engineering model provides a machine-independent execution environment for distributed applications.

Unlike the enterprise, information, and computational models which deal with the semantics of distributed applications, the engineering model is not concerned with the semantics of the distributed application, except to determine its requirements for distribution.

5.1 Engineering Model: An Object-Based Distributed Platform

The ODP engineering model is an architectural framework for the provision of an object-based distributed platform. The set of basic services and mechanisms, identified in the engineering model, are modelled as a collection of interacting objects which together provide support for the *realization* of interactions between distributed application components.

The engineering model can be considered as an extended operating system spanning a network of interconnected computers. In the *networked-operating system* view of the model, the linked computers preserve much of their autonomy and are managed by their local operating systems which are enhanced with mechanisms to enable, regulate and (if desired) hide distribution.

5.2 Engineering Model: Animation of Computational Model

The interest of the computational model is directly related to the existence of a mapping enabling it to relate to engineering concerns. This means, for instance, being able to map computational concepts onto the engineering structures.

The engineering model provides an infrastructure or a distributed platform for the support of the computational model. The model provides generic services and mechanisms capable of supporting distributed applications specified in the computational model. The model is concerned with *how* an application, specified in the computational model, may be *engineered* onto the distributed platform. The selection of distribution transparency and communication (protocol) objects, among many other support mechanisms, tailored to application needs, forms an important task.

The engineering model identifies the *functionality* of basic system components that must be present, in some form or other, in order to support the computational model. Hypothetically, there may be several engineering models for a particular computational environment, reflecting the use of different system components and mechanisms to achieve the same end. The issue in the computational model is *what* (interactions, distribution requirements); the engineering model prescribes solution as to *how* to realize these interactions, satisfying the stated requirements.

5.3 Structure of Engineering Model

The engineering model reveals the structure of the distributed platform, the ODP infrastructure which supports the computational model. The services or mechanisms which enable, regulate and hide distribution in the ODP infrastructure, are modelled as objects, called *engineering objects*, which may support multiple interfaces.

There are different kinds of engineering objects in the engineering model corresponding to different distribution (enabling, regulating, hiding) functions required in distributed environment. This is illustrated in Figure 4. Some engineering objects correspond to the application functionality and are referred to as *basic engineering objects* while those which provide distribution functions are classified as *transparency objects*, *protocol objects*, *support objects*, etc. At a given host, the basic engineering objects belonging to an application may be grouped into *clusters*. A host may support multiple clusters in its addressing domain, known as *capsule.* A capsule consists of clusters of basic engineering objects, set of transparency objects, protocol objects and other local operating system facilities.

From an engineering viewpoint, the ODP infrastructure consists of interconnected autonomous computer systems (hosts), which are called *nodes*. Each node supports a *nucleus object* and multiple capsules. The nucleus encapsulates computing, storage, and communication resources at a node. All the objects in the node share common processing, storage, and communication resources encapsulated in the nucleus object of the node.

As mentioned before, the engineering model *animates* the computational model. The computational-level interactions between a pair of computational objects (or their interfaces) are supported through *channel* structures in the engineering model. A channel binds basic engineering objects in different clusters, capsules, or nodes. The channel is a configuration of transparency objects, protocol objects, etc. which provide distribution support.

The services and mechanisms currently identified in the engineering model are generic in nature and can support distribution requirements of applications in a broad range of enterprise domains (Telecoms, Office Information Systems, Computer Integrated Manufacturing, etc.). However, domain-specific supporting functions will be defined in the domain-specific engineering models (which are the specialization of ODP engineering model).

The following is a brief description of the engineering objects and structures currently identified in the ODP engineering model. The objects and structures which are defined later in the text are italicized. Table 2 gives a relationship between the engineering objects and the real world system.

Basic Engineering Object: Basic Engineering Objects (BEOs) are the run time representation of computational objects (obtained through compilation, interpretation or through some other transformation of computational objects) which encapsulate application functionality.

Cluster: A cluster is a configuration of basic engineering objects. Clusters are used to express related objects (which belong to the same application) that should be local to one another, i.e., those groups of objects that should always be on the *same* node at all times.

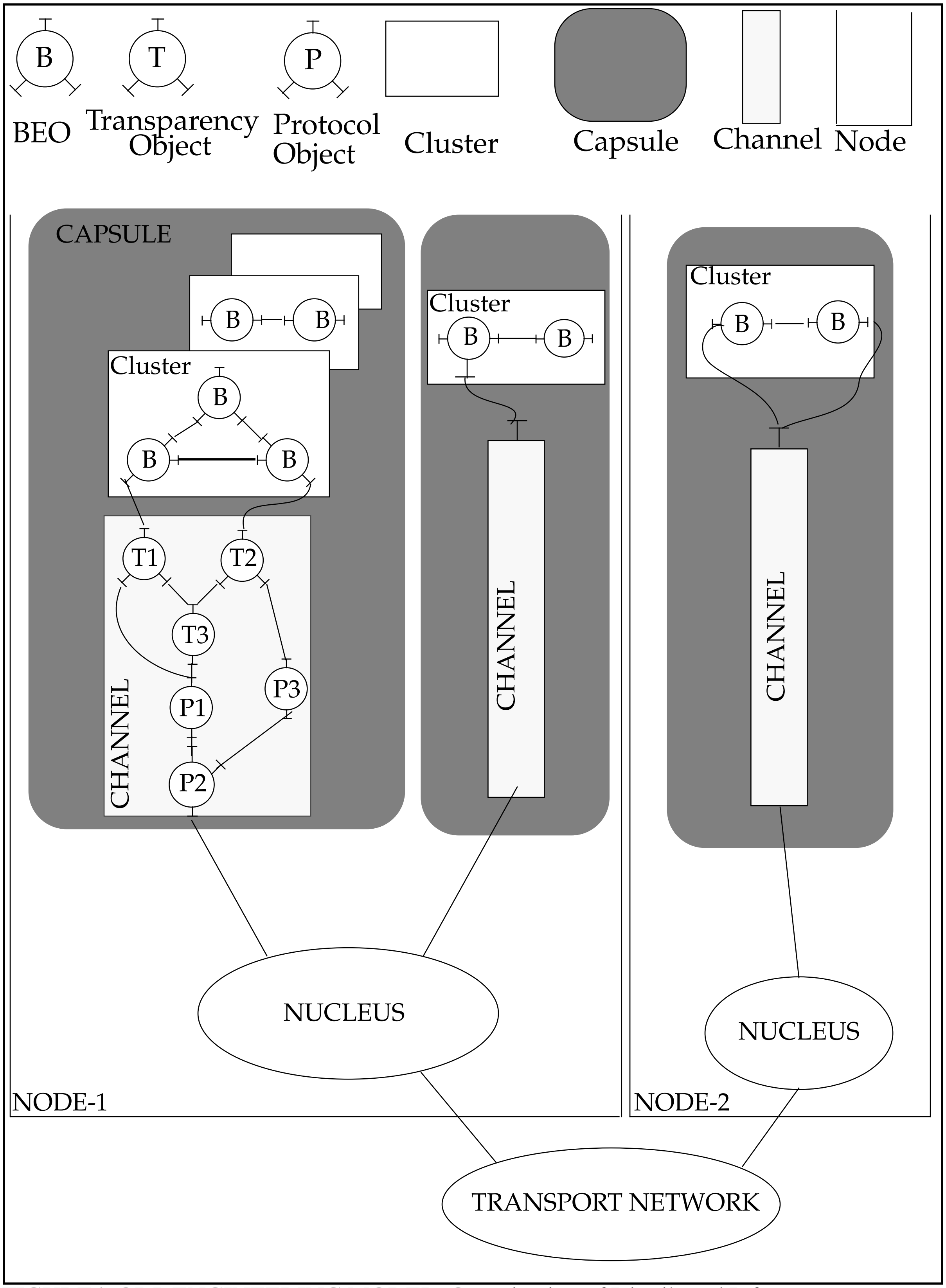

FIGURE4. ODP ENGINEERING MODEL: Organization of Distributed Infrastructure

Capsule: A capsule consists of clusters of basic engineering objects, *transparency objects*, and *protocol objects* bound to a common *nucleus* in a distinct address space from any other capsule. A capsule provides to its clusters access to the objects in the *channel* and to the nucleus to which it is bound.

Nucleus: A nucleus is an object that provides access to basic processing, storage, and communication functions of a *node* for use by basic engineering objects, *transparency objects, protocol objects,* bound together into capsules. A nucleus may support more than one capsule. A nucleus has the capability of interacting with other nuclei (through its communication function), providing the basis for inter-capsule and inter-node communication.

Node: A node consists of one nucleus object, a node manager, and a set of capsules. All of the objects in a node share common processing, storage, and communications resources.

Channel: A channel is a configuration of *transparency objects*, *protocol* objects, *application specific supporting objects*, etc. providing a binding between a set of interfaces to basic engineering objects, through which interaction can occur. The structure of the channel is dependent on the distribution function requirements of the interaction between basic engineering objects.

Table 2: System Abstractions in Engineering Model

Engineering object	System representation
Node	single computer system, network of workstations managed by a distributed operating system, any autonomous information processing system with independent *nucleus* resources and failure characteristics.
Nucleus	abstraction of an operating system providing processing, storage, and communication resources of a *node*.
Capsule	the concept of address space in operating systems.
Cluster	the concept of ‘linked’ modules to form an executable program image.
BEO	the program module which may not be executed in isolation.
Channel	the run time ‘binding’ between distributed BEOs
Transparency object	Special purpose modules which enhance the operating system environment of the *node* and can be dynamically linked into the distributed application program.

Figure 5 shows the client-half and server-half of a single channel object. If the objects being bound are on different nodes, there is still conceptually only one channel object created, i.e., there is not one channel object on one node and a different channel object on the other.

Stub Object: An object which acts to a basic engineering object as a *representative* of another basic engineering object located in different clusters, thus contributing towards distribution transparency. Stub objects are bound to the basic engineering objects for the purpose of hiding certain aspects resulting from distribution (or heterogeneity).

The stub objects have direct access to the basic engineering objects. The operation invocations on the interfaces of basic engineering objects are *intercepted* by stub objects to hide some aspects of distribution such as concurrency in the system or to modify the information exchanged between basic engineering objects, thus masking the heterogeneity in the distributed system.

Stub objects add further interactions and/or information to interactions between interacting basic engineering objects to support distribution transparency. As an example, a stub object may provide adaptation fu*cess tra*nctions such as marshalling and un-marshalling of openctions such asnctions such as marshalling and un-marshalling of operation parameters to enable *ac*ncnctions such as marshalling and un-marshalling of operation parameters to enable *ac*tionctions such as marshalling and un-marshalling of operation parameters to enable *ac*ns such as marshalling and un-marshalling of operation parameters to enable *ac* marshalling and un-marshalling of operation parameters to enable *ac*ration parameters to enable *acnsparent* interactions between interfaces of basic engineering objects.

Examples of stub objects include *access transparency object* and *concurrency transparency object* discussed in the next section.

Basic engineering objects are always directly bound to the stub objects. Stub objects within a channel can interact with one another using other objects in the channel, or via interaction with supporting objects outside of the channel.

Binder Object: An object which *controls* and *maintains* the binding between interacting basic engineering objects, contributing towards the provision of distribution transparency.

Binder objects maintain the binding between basic engineering objects, even if they are migrated, reactivated at new location, or are replicated. Examples of binder objects include *location transparency object, migration transparency object, replication transparency object, failure transparency object,* and *resource transparency objec*t.

Stub objects are bound to binder objects. Binder objects interact with one another to maintain the integrity of the binding between the interacting basic engineering objects. Binder objects in the channel can interact with one another using other objects in the channel, or via interaction with supporting objects outside the channel. Binder objects are interconnected by protocol objects.

Protocol Object: An object which encapsulates communication protocol functionality for supporting communication between basic engineering objects. A channel may be composed of a number of protocol objects corresponding to different communication support requirements of interactions between basic engineering objects. Protocol objects interact with other protocol objects to support interaction between basic engineering objects.

Interceptor Object: An object which masks administrative and technology domain boundaries by performing transformation functions such as protocol conversion, type conversion etc. It enables interactions to cross administrative and communication domains, thus contributing towards *federation transparency.*

Distribution Transparency: The following transparencies have been identified in the ODP engineering model, as important in distributed systems. The concept of transparency is viewed as the corner stone of ODP architecture. A brief description of transparencies, based on the concept of client and server objects (or client and server interfaces) is outlined below:

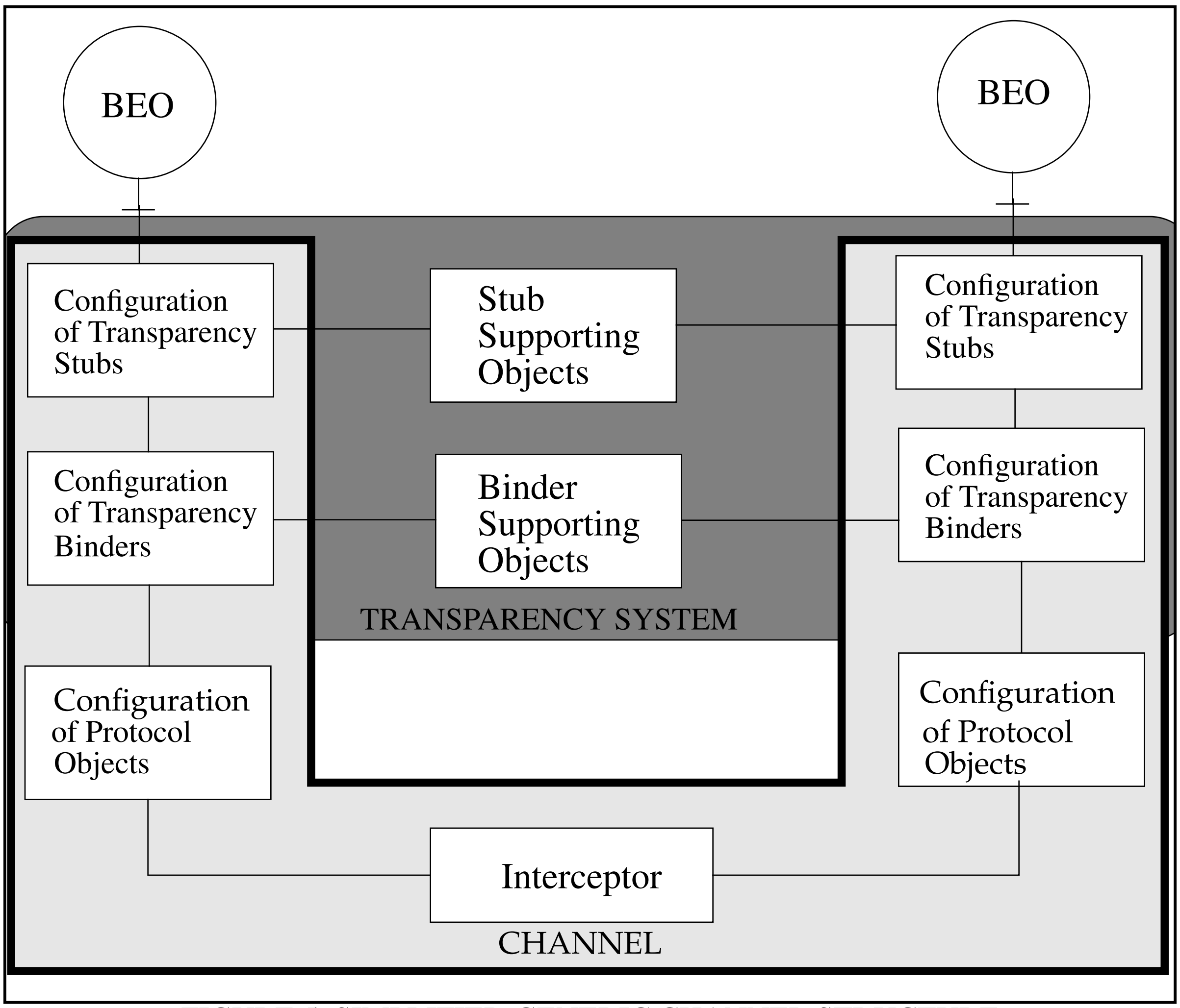

FIGURE 5. SIMPLIFIED GENERIC CHANNEL STRUCTURE

These transparency mechanisms provide an enhanced environment positioned on top of the low-level operating systems and communications facilities of the distributed platform, for the support of distribution transparent programming environment offered by the computational model.

The technique for providing any transparency service is based on the single principle of replacing an original service by a new service which combines the original service with the transparency service, and which permits clients to interact with it as if it were the original service. The clients need not be aware of how these combined services are achieved.

Since the interaction between the objects occur at their interfaces, these transparencies are applicable to individual interfaces or to specific operations of the interfaces. An interface may have a set of transparency requirements which may be different from those of other interfaces of the same object.

A summary of transparency mechanisms is presented in Table 3.

Access Transparency: It hides from a client object the details of the access mechanisms for a given server object, including details of data representation and invocation mechanisms (and vice versa). Access transparency hides the difference between local and remote provision of the service.

Access transparency enables interworking across heterogeneous computer architectures, operating systems and programming languages.

Concurrency Transparency: It hides from the client the existence of concurrent accesses being made to the server. Concurrency transparency hides the *effects* due to the existence of concurrent users of a service from individual users of the service.

Location Transparency: It hides from a user (client) where the object (server) being accessed is located.

Migration Transparency: Migration transparency hides from the user of the service (client) the effects of the provider of the service moving from one location to another, during the provision of the service (between successive operation invocations).

Location transparency is a static transparency in the sense that it is assumed that once located the interface remains at its location (until the binding between the involved interfaces is broken). Migration transparency is the dynamic case which arises if the server interface can move while the client object is interacting with it, without disturbing those interactions.

Replication Transparency: Replication transparency, also known as *group transparency,* hides the presence of multiple copies of services and maintaining the consistency of multiple copies of data, from the users of the services.

It enables a set of objects (their interfaces) organized as a *replica group* to be coordinated so as to appear to interacting objects (or their interfaces) as if they were a single object (interface).

There are two main aspects of replication transparency. The first hides the difference between a replicated and a non-replicated provider of a service from users of that service, and the second hides the difference between replicated and non-replicated users of a service from providers of that service.

Users are unaware of multiple providers of the service and need not concern about making multiple operation invocation or dealing with multiple responses.

Resource Transparency: It hides from a user (client) the mechanisms which manage allocation of resources by activating or passivating (server) objects as demand varies. It also implies the hiding of deactivation and reactivation of (server) objects from the clients.

Resource transparency, also known as *liveness transparency*, masks the automated transfer of clusters from a capsule to a storage object and back again, to optimize the use of node's nucleus resources (processor, memory, etc.).

With resource transparency in place, clients can invoke operations on the server irrespective of whether the server is currently active or passive.

Failure Transparency: Failure transparency masks (certain) failure(s) and possible recovery of server objects from the client objects, thus providing fault tolerance.

Federation Transparency: Federation transparency hides the effects of operations crossing multiple administrative boundaries from the clients. It permits interworking across multiple administration and technology domains.

Table 3: ODP Distribution Transparencies

Transparency	Central Issue	Result of Transparency
Access	The method of access to objects (invocation mechanism and data representation).	Client need not be unaware of *access* mechanisms at the server interface.
Concurrency	Concurrent access to objects in the distributed system.	Clients are masked from the effects of concurrent access to the server interface.
Location	Location of object in the distributed system.	Clients are unaware of the physical location of the server.
Migration	Dynamic re-location of objects during the "bind-session".	Clients are unaware of the dynamic migration of the server.
Replication	Multiple invocations on replicated objects, multiple responses, and consistency of replicated data.	Client invokes a replicated server group as if it were a single server. Distribution of requests, collation of responses, consistency of data, and membership changes are hidden.
Resource	Resource management policies of the *node* (deactivation and reactivation of objects).	Client unaware of the deactivation and reactivation of the server.
Failure	Partial failure of object in the *node*.	Client unaware of the failure of the server and its subsequent reactivation (possibly at another node).
Federation	Pan-organizational boundaries.	Clients unaware of interactions crossing administrative and technology boundaries.

6. CONCLUSION & SUMMARY OF PAPERS IN THE SPECIAL ISSUE

This introduction shall give the reader an overview of the principles of the reference model for Open Distributed Processing (ODP). Through the reference model, a framework is created to give descriptive and design support for distribution, interworking and portability. More information can be found in the four parts of the publicly available standardisation documents, i.e. ITU-T Recommendations X.901 to X.904. The standards however, must be populated by abstraction concepts, design and evaluation methods, engineering mechanisms etc. This special issue would like to make a contribution for the population of and the application of the ODP reference model.

One of the new challenges in the realm of ODP is the handling of multimedia and the communication by streams, as it is outlined in /Coul et al/. The integration of streams into an open distributed processing environment requires the development of new abstraction concepts, that allow the description of information streaming over time. Furthermore the question has been raised how to support the processing of streams from an engineering point of view. Audio and video streams need engineering support to maintain their isochronous nature over time. In order to maintain a distinguished quality of the streams, resource allocation and resource control mechanism is to be provided by the framework.

By the specification of a multiparty audio and video interaction the suitability of the RM-ODP concepts and rules as described by the five viewpoint languages has been evaluated in /Gay etal/. The audio-video interaction facilities are modelled in each viewpoint. In the enterprise model the requirements of the stream binding object are specified. The binding contract is an element of the information view. The bound objects exchanging video and audio streams are specified distribution-transparently by terms of the computational language. The distribution support for the computational model is an issue of the engineering viewpoint.

A model for the process of designing ODP systems is another item dealt with in /Sind etal/. The model comprises a mapping of ODP concepts and designing rules onto formal description constructs. Such a mapping is called architectural semantics, that allows a specification being interpreted by means of ODP concepts and rules.

The formal handling of ODP description concepts from part I and II of the reference model, and their relations to FDTs is discussed in /Gotz/ and /Najm etal/. It is part of the architectural semantics principle to provide with formal and their interpretation. Formal compositional notions of architectures, behaviours and systems are being introduced together with design concepts of abstraction and refinement. The latter ones are introduced in order to refine or abstract consistently from parts of an architecture, a behaviour or a system.

Whereas this is a more general approach to distributed system design, /Najm etal/ have restricted themselves to the introduction of a formal operational semantics for a language-independent modelling framework of the computational viewpoint. Type checking procedures and rewriting rules are introduced for the purpose of early consistency checks at the application of strongly typed object interfaces.

7. PAPERS IN THIS ISSUE

/ISO 10746-1 to 4/ ODP Reference Model Part I to IV 1994.

/Coul. etal/ G.Coulson, G.S.Blair Lancaster University GB, J-B.Stefani, F.Horn, L.Hazard CNET Paris
Supporting the Real-Time Requirements of Continuous Media in ODP

/Gay etal/ Valerie Gay Universite Paris VI, Peter Leydekkers PTT Research Groningen Netherlands, Robert Huis in't Veld University of Twente, Netherlands
Specification of Multiparty Audio and Video Interaction based on the RM-ODP

/Sind. etal/ Marten van Sinderen, Luis Ferreira Pires, Chris Vissers, Joost-Pieter Katoen University of Twente, Netherlands

/Gotz/ Reinhard Gotzhein Universität Kaiserslautern Germany
Towards a Basic Reference Model of Open Distributed Processing

/Najm et al./ Elie Najm ENST Paris, Jean-Bernard Stefani CNET Paris
A formal Semantics for the ODP Computational Model
CNIS Special Issue on ODP

8. SUGGESTED REFERENCES

1. Draft Recommendation ITU-T X.901 / ISO 10746-1: Basic Reference Model of Open Distributed Processing - Part-1: Overview.
2. Draft International Standard ITU-T X.902 / ISO 10746-2: Basic Reference Model of Open Distributed Processing - Part-2: Descriptive Model.
3. Draft International Standard ITU-T X.903 / ISO 10746-3: Basic Reference Model of Open Distributed Processing - Part-3: Prescriptive Model.
4. Draft Recommendation ITU-T X.904 / ISO 10746-4: Basic Reference Model of Open Distributed Processing - Part-4: Architectural Semantics.
5. Proceedings of the IFIP TC6/WG6.4 International Workshop on Open Distributed Processing (October 1991), North Holland 1992.
6. Proceedings of the International Conference on Open Distributed Processing (September 1993), Berlin.
7. Proceedings of the First Telecommunication Information Networking Architecture Workshop, (TINA 90), Lake Mohonk, New York, USA, June 1990.
8. Proceedings of the Second Telecommunication Information Networking Architecture Workshop, (TINA 91), Chantilly, France, March 1991.
9. Proceedings of the Third Telecommunication Information Networking Architecture Workshop, (TINA 92), Narita, Japan, January 1992
10. Proceedings of the Fourth Telecommunication Information Networking Architecture Workshop, (TINA 93), L'Aquila, Italy, September 1993.