# Reference Model of Open Distributed Processing (RM-ODP): Introduction

Kerry Raymond

kerry@dstc.edu.au
CRC for Distributed Systems Technology
Centre for Information Technology Research
University of Queensland
Brisbane 4072 Australia

**Abstract**

The Reference Model of Open Distributed Processing (RM-ODP) was a joint effort by the international standards bodies ISO and ITU-T to develop a coordinating framework for the standardisation of open distributed processing (ODP). The model describes an architecture within which support of distribution, interworking, interoperability and portability can be integrated. The RM-ODP framework defines ODP concerns using five "viewpoints" (abstractions), namely enterprise, information, computational, engineering, and technology. This tutorial introduces the reference model, describing the viewpoints and some of the ODP functions and transparencies.

Keyword Codes: C.2.4
Keywords: Computer-Communication Networks, Distributed Systems

## 1. WHAT IS RM-ODP?

Advances in computer networking have allowed computer systems across the world to be interconnected. Despite this, heterogeneity in interaction models prevents interworking between systems. Open distributed processing (ODP) describes systems that support heterogeneous distributed processing both within and between organisations through the use of a common interaction model.

ISO and ITU-T (formerly CCITT) have developed a Reference Model of Open Distributed Processing (RM-ODP) to provide a coordinating framework for the standardisation of ODP by creating an architecture which supports distribution, interworking, interoperability and portability.

### 1.1. The Goals and Deliverables of RM-ODP

RM-ODP aims to achieve:

- portability of applications across heterogeneous platforms
- interworking between ODP systems, i.e. meaningful exchange of information and convenient use of functionality throughout the distributed system
- distribution transparency, i.e. hide the consequences of distribution from both the applications programmer and user

The reference model provides a "big picture" that organises the pieces of an ODP system into a coherent whole. It does not try to standardise the components of the system nor to unnecessarily influence the choice of technology.

There are many challenges in developing a reference model. RM-ODP must be adequate to describe most "reasonable" distributed systems available both today and in the future, so RM-ODP is abstract, but not vague. RM-ODP carefully describes its components without prescribing an implementation.

### 1.2. Structure of RM-ODP

The RM-ODP standard is known as both ISO International Standard 10746 and ITU-T X.900 Series of Recommendations and will consist of four parts:

- Part 1: Overview and Guide to Use (ISO 10746-1/ITU-T X.901) [1]
- Part 2: Descriptive Model (ISO 10746-2/ITU-T X.902) [2]
- Part 3: Prescriptive Model (ISO 10746-3/ITU-T X.903) [3]
- Part 4: Architectural Semantics (ISO 10746-4/ITU-T X.904) [4]

Part 1 contains a motivational overview of ODP and explains the key concepts of the RM-ODP architecture. Part 2 gives precise definitions of the concepts required to specify distributed processing systems. Part 3 prescribes a framework of concepts, structures, rules, and functions required for open distributed processing. Part 4 describes how the modelling concepts of Part 2 can be represented in a number of formal description techniques.

This tutorial focuses on the framework established in Part 3.

### 1.3. Status of RM-ODP

In January 1995, Parts 2 and 3 of RM-ODP were successfully balloted to become International Standards; official copies of the standard will be soon be available, after the necessary administrative processing.

Parts 1 and 4 are based on Parts 2 and 3. Therefore, the standardisation of Parts 1 and 4 follows the standardisation of Parts 2 and 3. Parts 1 and 4 are currently Committee Drafts and expected to become Draft International Standards in April 1995 and International Standards in early 1996.

## 2. VIEWPOINTS

Part 3 of RM-ODP prescribes a framework using *viewpoints* from which to abstract or view ODP systems. A set of concepts, structures, and rules is given for each of the viewpoints, providing a "language" for specifying ODP systems in that viewpoint.

RM-ODP defines the following five viewpoints:

- Enterprise Viewpoint (purpose, scope and policies)
- Information Viewpoint (semantics of information and information processing)
- Computational Viewpoint (functional decomposition)
- Engineering Viewpoint (infrastructure required to support distribution)
- Technology Viewpoint (choices of technology for implementation)

Specifying an ODP system using each of the viewpoint languages allows an otherwise large and complex specification of an ODP system to be separated into manageable pieces, each

focused on the issues relevant to different members of the development team. For example, the information analyst works with the information specification while the systems programmer is concerned with the engineering viewpoint. Figure 1 shows how the RM-ODP viewpoints can be related to the software engineering process.

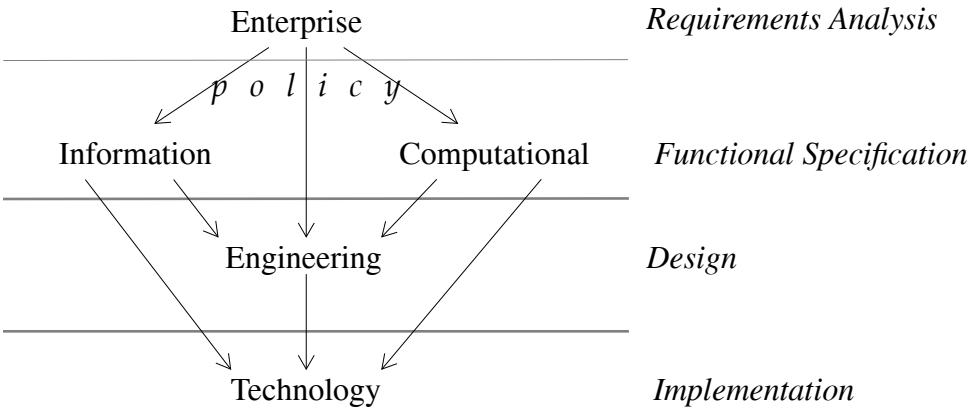


Figure 1: RM-ODP Viewpoints and Software Engineering

## 3. ENTERPRISE VIEWPOINT

The enterprise viewpoint is used to organisational requirements and structure. In the enterprise viewpoint, social and organizational policies can be defined in terms of:

- objects — both "active" objects, e.g. bank managers, tellers, customers, and "passive" objects, e.g. bank accounts, money
- communities — groupings of objects intended to achieve some purpose, e.g. a bank branch consists of a bank manager, some tellers, and some bank accounts; the branch provides banking services to a geographical area
- roles of the objects within communities, expressed in terms of policies:
  * permission — what can be done, e.g. money can be deposited into an open account
  * prohibition — what must not be done, e.g. customers must not withdraw more than $500 per day
  * obligations — what must be done, e.g. the bank manager must advise customers when the interest rate changes

The enterprise language is specifically concerned with *performative actions* that change policy, such as creating an obligation or revoking permission. In a bank, the changing of interest rates is a performative action as it creates obligations on the bank manager to inform the customers. However, obtaining an account balance is not a performative action as obligations, permissions, and prohibitions are not affected. Thus, an enterprise specification of a bank need not include the obtaining of account balances; such functionality will be identified in the computational specification.

By preparing an enterprise specification of an ODP application, policies are determined by the organisation rather than imposed on the organisation by technology (implementation) choices. For example, a customer should not be limited to having only one bank account, simply because it was more convenient for the programmer.

## 4. INFORMATION VIEWPOINT

The information viewpoint is used to describe the information required by an ODP application through the use of schemas, which describe the state and structure of an object; e.g., a bank account consists a balance and the "amount withdrawn today".

A *static schema* captures the state and structure of a object at some particular instance; e.g., at midnight, the amount-withdrawn-today is $0.

An *invariant schema* restricts the state and structure of an object at all times; e.g., the amount-withdrawn-today is less than or equal to $500.

A *dynamic schema* defines a permitted change in the state and structure of an object; e.g. a withdrawal of $X from an account decreases the balance by $X and increases the amount-withdrawn-today by $X. A dynamic schema is always constrained by the invariant schemas. Thus, $400 could be withdrawn in the morning but an additional $200 could not be withdrawn in the afternoon as the amount-withdrawn-today cannot exceed $500.

Schemas can also be used to describe relationships or associations between objects; e.g., the static schema "owns account" could associate each account with a customer.

A schema can be composed from other schemas to describe complex or composite objects; e.g., a bank branch consists of a set of customers, a set of accounts, and the "owns account" relationships.

The information specification of an ODP application could be expressed using a variety of methods, e.g., entity-relationships models, conceptual schemas, and the **Z** formal description technique.

## 5. COMPUTATIONAL VIEWPOINT

The computational viewpoint is used to specify the functionality of an ODP application in a distribution-transparent manner. RM-ODP's computational viewpoint is object-based, that is:

- objects encapsulate data and processing (i.e. behaviour)
- objects offer interfaces for interaction with other objects
- objects can offer multiple interfaces.

A computational specification defines the objects within an ODP system, the activities within those objects, and the interactions that occur among objects. Most objects in a computational specification describe application functionality, and these objects are linked by bindings through which interactions occur. Binding objects are used to describe complex interaction between objects.

Objects in a computational specification can be application objects (e.g. a bank branch) or ODP infrastructure objects (e.g. a type repository or a trader, see Section 8.3.1 and Section 8.3.2). Figure 2 illustrates a bank branch object providing a bank teller interface and a bank manager interface. Both interfaces can be used to deposit and withdraw money, but accounts can be created only through the bank manager interface. Each of the bank branch object's interfaces is bound to a customer object.

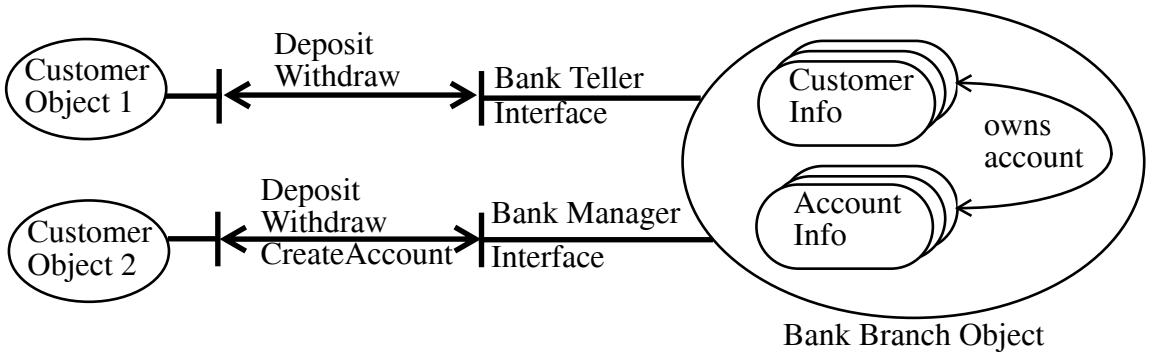


Figure 2: Bank Branch Object with Bank Manager and Bank Teller Interfaces

### 5.1. Computational Interaction

RM-ODP provides three forms of interaction between objects: operational, stream-oriented, and signal-oriented.

Operational interfaces provide a client-server model for distributed computing—client objects invoke operations at the interfaces of server objects (i.e. the remote procedure call paradigm). Operational interfaces consist of named operations with parameters, terminations, and results. Operations in RM-ODP can be either interrogations (which return a termination) or announcements (which do not return a termination).

For example, a bank branch object offers a number of BankTeller operational interfaces, whose signature is defined as:

```
BankTeller = Interface Type {

  operation Deposit (c: Customer, a: Account, d: Dollars)
    returns OK (new_balance: Dollars)
    returns Error (reason: Text);

  operation Withdraw (c: Customer, a: Account, d: Dollars)
    returns OK (new_balance: Dollars)
    returns NotToday (today: Dollars, daily_limit: Dollars)
    returns Error (reason: Text);
}
```

Note that the notation used in the example above is merely illustrative. RM-ODP does not prescribe any particular notation for defining operational interface types.

Stream interfaces provide (logically) continuous streams of information flowing between producer and consumer objects. Consumer objects connect to the stream interfaces of producer objects or vice-versa, and several streams can be grouped in a single interface, e.g., an audio stream and a video stream. Stream interfaces have been included in RM-ODP to cater for multi-media and telecommunications applications.

Underlying both operational interfaces and stream interfaces are signal interfaces which provide very low-level communications actions. The OSI service primitives (REQUEST, INDICATE, RESPONSE, and CONFIRM) are examples of signals.

### 5.1.1. Interface Subtyping

The concept of interface type is particularly important in RM-ODP. Interfaces in the computational model are strongly typed and inheritance of an interface type (usually) creates a subtype relationship. Subtypes of an interface type are substitutable for the parent type (or any supertype).

Figure 3 illustrates interface subtyping.The BankManager and LoansOfficer interface types are subtypes of the BankTeller interface (super-)type; either can substitute for a BankTeller as they can perform the Deposit and Withdraw operations expected of a BankTeller. Neither a BankTeller nor a LoansOfficer can replace a BankManager, as neither can provide the CreateAccount operation.

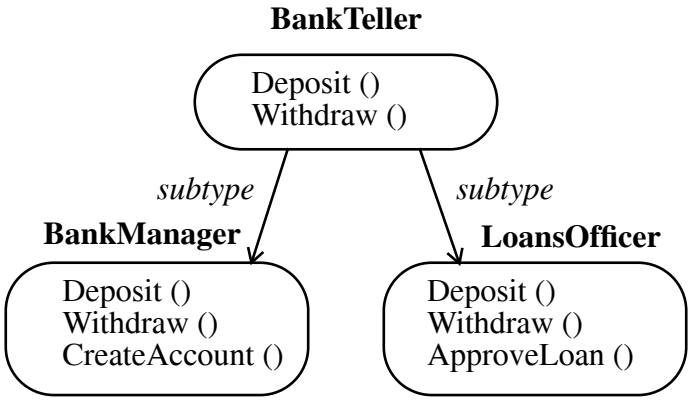


Figure 3: Example of Interface Subtyping

## 5.2. Computational Activity

The computational viewpoint also defines the actions that are possible within a computational object. These are:

- creating and destroying an object
- creating and destroying an interface
- trading for a interface (see Section 8.3.2)
- binding to an interface
- reading and writing the state of the object
- invoking an operation at an operational interface
- producing/consuming a flow at a stream interface
- initiating or responding to a signal at a signal interface.

These basic actions can be composed in sequence or in parallel. If composed in parallel, the parallel activities can be dependent (the activity is forked and must subsequently join at a synchronisation point) or independent (the activity is spawned and cannot join).

### 5.3. Environment Contracts

The refinement of a computational object and its interfaces might require the specification of requirements on the realization of that object or its interfaces (and, hence, of the objects with which it interacts). For example, a bank must protect the customer's money and must ensure that interaction is secure against a variety of fraudulent activities, e.g. capturing and replaying operations. Therefore, the actual interactions must either be communicated over a secure network or employ end-to-end security checks.

Ideally, environment contracts will be expressed in high-level quality-of-service terms rather than, e.g., specifying a particular network or a particular encryption scheme (either of which presupposes the environment in which the ODP system will operate).

Currently, the state of the art falls short of this ideal. However, it is important that RM-ODP be "future-proof", capable of incorporating both current and expected future technologies.

## 6. ENGINEERING VIEWPOINT

The engineering viewpoint is used to describe the design of distribution-oriented aspects of an ODP system; it defines a model for distributed systems infrastructure. The engineering viewpoint is not concerned with the semantics of the ODP application, except to determine its requirements for distribution and distribution transparency.

The fundamental entities described in the engineering viewpoint are objects and channels. Objects in the engineering viewpoint can be divided into two categories—basic engineering objects (corresponding to objects in the computational specification) and infrastructure objects (e.g., a protocol object — see below). A channel corresponds to a binding or binding object in the computational specification.

### 6.1. Channels

A channel provides the communication mechanism and contains or controls the transparency functions required by the basic engineering objects, as specified in the environment contracts in the computational specification. Figure 4 illustrates the channel between a Customer Object and the Bank Branch object in Figure 2. The shaded area is the channel, composed of stubs, binders, and protocol objects. Stubs and binders are used to provide various distribution transparencies.

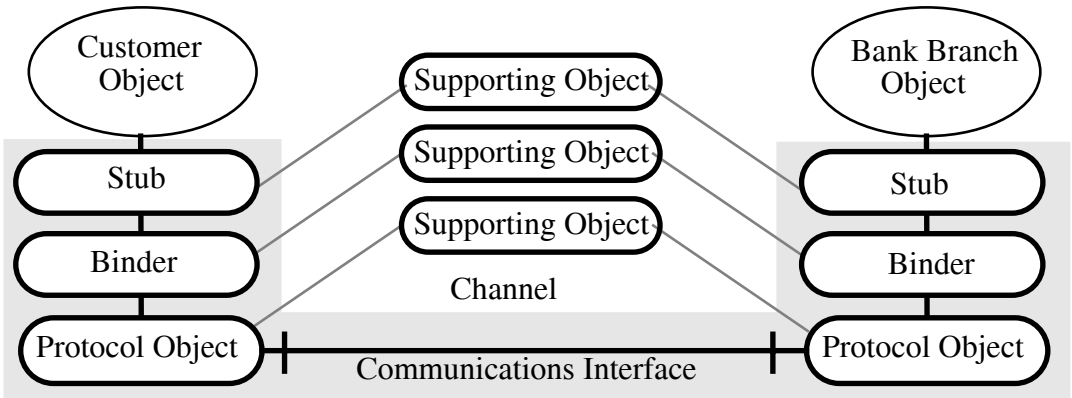


Figure 4: Structure of a Channel

Stubs are used when the transparency involves some knowledge of the application semantics, e.g., maintaining a log of operations for an audit trail.

Binders are used when application semantics are not required; they merely transport the messages (bit streams). Binders are responsible for managing the binding between the basic engineering objects; e.g., binders could use sequence numbers to foil capture-and-replay attempts.

Protocol objects interact via a communications interface; this models networking.

Outside of the channel, supporting objects assist the stub, binder, and protocol objects within the channel. Typically, supporting objects are repositories of information required by the stubs, binders, and protocol objects. For example, binders register and retrieve interface locations via a supporting object known as the relocator (see Section 8.3.3) in order to achieve location transparency.

### 6.2. Engineering Structures

The RM-ODP engineering viewpoint prescribes the structure of an ODP system. The basic units of structure are:

- cluster — a set of related basic engineering objects that will always be co-located
- capsule — a set of clusters, a cluster manager for each cluster, a capsule manager, and the parts of the channels which connect to their interfaces
- nucleus object — an (extended) operating system supporting ODP
- node — a computer system

Figure 5 illustrates the structure of a node.

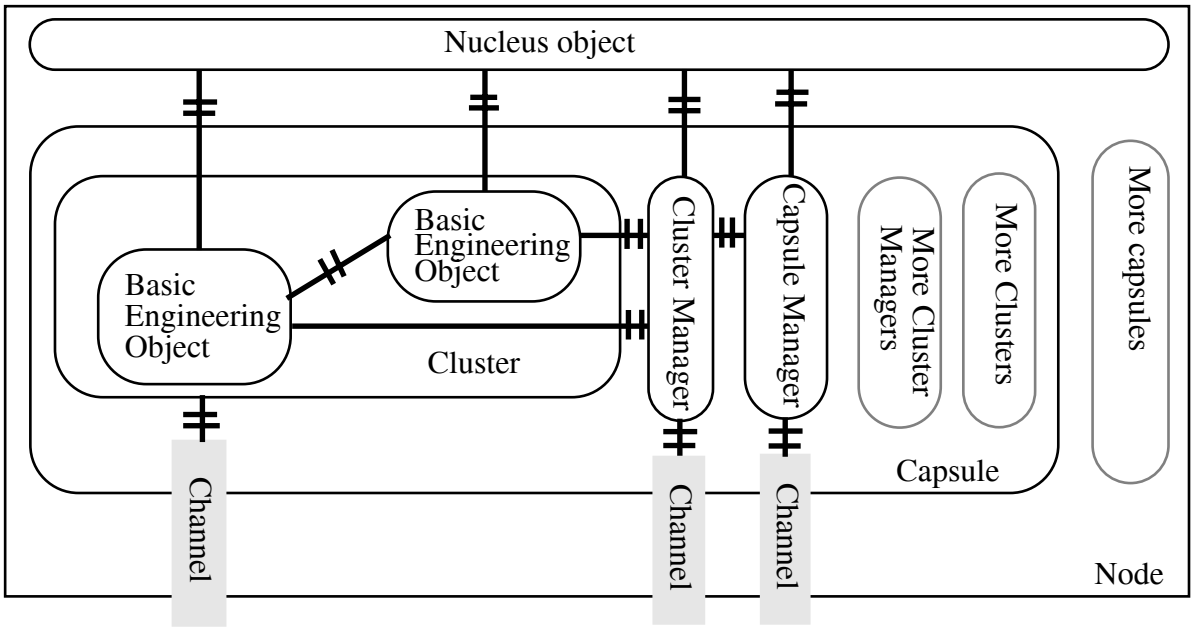


Figure 5: Structure of a Node

Given these definitions, the following structuring rules are defined:

- a node has a nucleus object
- a nucleus object can support many capsules
- a capsule can contain many clusters
- a cluster can contain many basic engineering objects
- a basic engineering object can contain many activities
- all inter-cluster communication is via channels

An implementation of an ODP system can choose to constrain the structuring, for example, by allowing:

- only one object per cluster
- only one cluster per capsule

## 7. TECHNOLOGY VIEWPOINT

A technology specification of an ODP system describes the implementation of that system and the information required for testing. RM-ODP has very few rules applicable to technology specifications.

## 8. ODP FUNCTIONS

The ODP functions are a collection of functions expected to be required in ODP systems to support the needs of the computational language (e.g. the trading function) and the engineering language (e.g. the relocator). The following subsections outline the major function groups in RM-ODP; a few of the functions are discussed in more detail to illustrate the scope of RM-ODP.

### 8.1. Management Functions

RM-ODP defines a number of functions to manage the engineering structures, including:

- node management function (provided by the nucleus) for creating capsules and channels
- capsule management function (provided by the capsule manager) for instantiating clusters and checkpointing and deactivating clusters in a capsule
- cluster management function (provided by the cluster manager) for checkpointing, deactivating and migrating clusters
- object management function (provided by the basic engineering object) for checkpointing and deleting basic engineering objects

### 8.2. Coordination Functions

RM-ODP defines a number of functions aimed at coordinating the actions of a number of objects, clusters, or capsules in order to produce some consistent overall effect. These include:

- checkpoint and recovery
- deactivation and reactivation
- event notification
- groups and replication
- migration
- transactions

### 8.2.1. Transaction Function

In the information viewpoint, state change appears to happen as a single indivisible action. However, in a computational and engineering viewpoints, this state might be distributed throughout the ODP system and be concurrently accessed by many parallel activities. In order to develop reliable ODP systems, it will be necessary to coordinate the behaviour of objects to achieve the desired degrees of:

- visibility — the degree to which the intermediate effects of an operation (or other interaction) are visible to other operations
- recoverability — the state after the failure of the operation (which of its effects are undone?)
- permanence — the consequences of the failure of the operation on completed operations (are their effects altered?)

RM-ODP defines a very generalised transaction function; this is another example of "future-proofing" in RM-ODP. Realistically, the ACID transaction model will be the only style of transaction mechanism supported by most ODP systems for a number of years. Consequently, RM-ODP defines an ACID transaction function as specialisation of its generalised transaction function.

## 8.3. Repository Functions

In addition to a general storage function and a general relationship repository, RM-ODP defines a number of specific repository functions, concerned with maintaining a database of specialised classes of information.

### 8.3.1. Type Repository

In most computer systems, type definitions are not explicitly maintained within the system. Instead, types are documented in manuals or defined according to some local conventions (e.g. use of mnemonic file names). ODP systems must make type information available through the ODP system itself; the primary need is to support type checking during trading and interface binding.

In RM-ODP, the type repository is a registry for type definitions, particularly for interface types.The type registry maintains a type hierarchy (subtype relationships) and other relationships between types.

### 8.3.2. Trader

The ODP Trader provides "a dating service for objects"; its purpose is to support dynamic binding by allowing services to be discovered at run-time. The trader is a repository of service advertisements.

Server objects advertise their services through a trader; the service advertisement specifies the interface type and service attributes. Servers manipulate their service advertisements by using the *export* operations provided by the trader. Clients choose services by specifying the required type and attributes in *import* operations.

The ODP Trader is also the subject of standardisation, separate from RM-ODP. An introduction to this standard can be found in [5].

### 8.3.3. Relocator

The relocator is a repository of interface locations (a “white pages” service). This information is needed by relocation transparency (see Section 9.2).

## 8.4. Security Functions

RM-ODP defines a number of security functions (e.g. access control, authentication, auditing) based on OSI Security Frameworks in Open Systems [6].

# 9. TRANSPARENCIES

Computational specifications are intended to be distribution-transparent, i.e., written without regard to the very real difficulties of implementation within a physically distributed, heterogeneous, multi-organisational environment. The aim of transparencies is to shift the complexities of distributed systems from the applications developers to the supporting infrastructure.

RM-ODP defines a number of commonly required distribution transparencies and describes the computational refinements and use of engineering functions needed to provide these transparencies. The distribution transparencies defined in RM-ODP are:

- access transparency — hides the differences in data representation and procedure calling mechanism to enable interworking between heterogeneous computer systems
- location transparency — masks the use of physical addresses, including the distinction between local versus remote
- relocation transparency — hides the relocation of an object and its interfaces from other objects and interfaces bound to it
- migration transparency — masks the relocation of an object from that object and the objects with which it interacts
- persistence transparency — masks the deactivation and reactivation of an object
- failure transparency — masks the failure and possible recovery of objects, to enhance fault tolerance
- replication transparency — maintains consistency of a group of replica objects with a common interface
- transaction transparency — hides the coordination required to satisfy the transactional properties of operations

The transparencies defined in RM-ODP are not intended to be the complete set, merely a starting point of common requirements. Additional transparencies for both general and specific needs could be subsequently standardised. For example, lip-sync transparency could be defined for stream interfaces supporting audio-visual interaction.

The following subsections describe some of the RM-ODP transparencies in more detail.

## 9.1. Access Transparency

Access transparency hides the differences in data representation and operation invocation from communicating objects.

Access transparency can be achieved by configuring the channel with stubs, which translates the desired interaction (e.g. an operation invocation) into a sequence of messages sent over a channel. The stubs must marshal and unmarshal any data used in the interaction in order to convert between different representations.

### 9.2. Relocation Transparency

Relocation transparency frees a basic engineering object (and the programmer of the object) from needing to know if an interacting object is relocated.

Relocation transparency can be achieved by configuring the channel with binders, which:

- inform the relocator (see Section 8.3.3) of the location of the interface it supports
- obtain from the relocator the location(s) of the other interface(s) connected to the channel

Binders will typically cache location information. If the location of an interface changes, the use of the old location will cause an error. With relocation transparency, the binder will automatically obtain the new location from the relocator, reconnect the channel, and replay the interaction. The basic engineering object should remain unaware of the change in location.

### 9.3. Transaction Transparency

Unlike access and relocation transparency which are achieved through configuring engineering channels with clever components, transaction transparency cannot be achieved by this mechanism alone.

The correct operation of the transaction function requires the reporting of the execution (or undo-ing) of certain "actions of interest" (e.g. reading or writing a piece of transaction-managed data). These events occur internal to the objects and are not visible to a stub or binder configured in the channel. Therefore, transaction transparency must involve the refinement of a transaction-transparent specification into a specification which reports the execution of these actions of interest to the transaction function.

## 10. SUMMARY

RM-ODP is a reference model, not an implementation standard; it defines a framework for the standardisation of open distributed processing. The RM-ODP model defines five viewpoints which decompose the specification of ODP applications by focusing on separate concerns.

The enterprise viewpoint defines a model for policy analysis while the information viewpoint provides a model for information analysis. The computational viewpoint defines a model for distributed programming languages; the run-time support for these languages is provided by the distributed systems infrastructure based on the engineering viewpoint model and the ODP infrastructure functions. The technology viewpoint is used to describe implemented systems.

## ACKNOWLEDGEMENTS

The author thanks Andrew Berry for his assistance in preparing this paper and all of the participants in the Australian and international RM-ODP standards groups for the many hours of lively discussions.

The participation of the author in the standardisation of RM-ODP has been supported by:

- Telecom (Australia) Research Laboratories through the Centre of Expertise in Distributed Information Systems (CEDIS)
- the Cooperative Research Centre for Distributed Systems Technology through the Cooperative Research Centres Program of the Department of the Prime Minister and Cabinet of the Commonwealth Government of Australia
- Standards Australia through their travel assistance scheme.

## REFERENCES


[1] ISO/IEC CD 10746-1, “Basic Reference Model of Open Distributed Processing - Part 1: Overview and Guide to Use”, July 1994.

[2] ISO/IEC DIS 10746-2, “Basic Reference Model of Open Distributed Processing - Part 2: Descriptive Model”, February 1994.

[3] ISO/IEC DIS 10746-3, “Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model”, February 1994.

[4] ISO/IEC CD 10746-4, “Basic Reference Model of Open Distributed Processing - Part 4: Architectural Semantics”, July 1994.

[5] M.Y. Bearman, “ODP-Trader”, International Conference on Open Distributed Processing, Berlin, September 1993.

[6] ISO/IEC CD 10181, “Security Frameworks in Open Systems”.