HeCSE Workshop on Emerging Technologies in Distributed Systems
Lammi, January 12-13, 1998

Architectures for Distributed Systems: Open Distributed Processing Reference Model

Lea Kutvonen

Department of Computer Science, University of Helsinki

Abstract

In this paper, we study the contents and role of a standardised framework for open distributed processing (RM-ODP). The reference model introduces some communication related concepts that are fundamental for the development of modern, global information services. Open platforms usually miss some of these concepts and thus induce difficulties for service development.

Keywords: open system frameworks, ODP reference model, object models

1 Introduction

Open systems are commonly understood as "a comprehensive and consistent set of international information technology standards and functional standards that specify interfaces, services and supporting formats to accomplish interoperability and portability of applications, data and people" [4]. In respect to a hierarchical machine model of computer systems, this requirement is normally considered to apply to the topmost abstraction layer. The solutions supporting such open interfaces are usually hidden, in order to support application software portability and distribution transparency of communication. Depending on the system purpose, the abstraction level of the open interface varies. Examples of open interfaces include

- parallel programming environment interfaces that tolerate different processor architectures or memory organisations,
- remote procedure call interfaces that hide transformations of data representation formats locally used in communicating computers,
- middleware system interfaces that hide differences of operating system services, and
- federated database management system interfaces that hide the differences of different information models in the member databases.

Currently, openness is usually expected from the middleware layer services. Systems like DCE [23], CORBA [22], and TINA DPE of TINA [3] are considered to fulfil this requirement. Middleware supports a consistent platform interface on which distributed applications can be easily implemented. However, interoperation between various platform implementations create problems, because of the variance in their behaviour.

When we try to create world-wide application services, we need to create interoperation facilities between sovereign systems. A sovereign system is an independently administered computing system, that includes autonomously selected services, interfaces, and uses autonomously selected languages, protocols, and information representation techniques. Interoperation between such sovereign systems can be established only via negotiations about commonly available facilities, because inherited similarity of technology or behaviour can not be expected.

Therefore, open systems must in addition of using published interfaces, also be able to exchange information about interfaces and the service behaviour available through those interfaces. Such facilities are defined in the family of standard for open distributed processing, the ODP reference model.

The ODP reference model (RM-ODP) [9, 16, 19, 24, 1] is a joint standardisation effort of ISO and ITU. It was started with a basic reference model standardisation officially already in 1989, and developed in interaction with other distributed system models. Thus, ODP reference model has had impact on industry trends already during its development.

The ODP standardisation aims for development of standards that allow distributed information processing systems to be exploited in a heterogeneous environment and under multiple organisational domains [10]. This goal is an enhancement to the openness requirement above. In addition to the use of public and standardised middleware services, the systems must be able to support interoperation in spite the independent evolution and independent technology decisions made by the sovereign member systems.

The ODP standards support systems to be built so that they

- provide software portability and interoperability;
- support integration of various systems with different architectures and resources without costly ad-hoc solutions;
- accommodate system evolution and run-time changes;
- federate across autonomously administered or technically differing domains;
- incorporate quality of service aspects to failure concepts;
- include security service; and
- offer selectable distribution transparency services for communication [10].

These goals are acquired by the ODP reference model through three already standardised aspects of the basic reference model (which that was completed in 1996 [12, 11]):

- a division of an ODP system specification into viewpoints, in order to simplify the description of complex systems [11];
- a set of general concepts for expressing the viewpoint specifications [12]; and
- a model for an infrastructure supporting, through the provision of distribution transparencies, the general concepts that it offers for specification purposes [11].

The next step in ODP standardisation work is to specify essential middleware services as component standards. These services include the already completed trading service [13, 17] and naming framework [5]. Components that are currently under work include type repository function [14], and interface binding framework [6] together with the supporting protocols [7]. At the same time, an open system management architecture (ODMA) [8] is being developed (although the current concepts are still more OSI related than ODP related). Further development is planned on the viewpoint languages specified by the basic reference model. First of the languages, enterprise language, is already being drafted.

The ODP reference model is a general standard that supports more specific frameworks at several application domains. For instance, TINA architecture can be seen as a telecommunication specific refinement. Consortia like OMG (founded in 1989) and TINA-C (founded in 1992) have formed liaisons with the ISO and ITU groups working on the standards, and have already adopted the trading functionality, and some of the vocabulary. The abstract infrastructure model serves as a long-term prediction of the joint software market in future.

The rest of the paper is organised as follows. Chapters 2, 3, and 4 study the basic concepts of the ODP framework. The concepts of transparency are introduced in Chapter 5, and the viewpoint languages in Chapter 6. Some of the open platform functions are sketched in Chapter 7. As a conclusion, we summarise the contributions of the ODP reference model for distributed systems.

2 ODP object model

The ODP object model differs from object models represented for object-oriented programming languages, object-oriented design methodologies, and distributed object management systems. The major differences include

- the flexible use of the object concepts for a variety of abstractions, ranging from implementation details to abstract business model considerations,
- the use of multiple interfaces,
- separation of client and server sides of the interface through which the objects communicate,
- variety of mechanisms for introducing objects to a system,
- concepts related to the type safety of communication through an interfaces, and

- interactions between objects are not constrained – they can be asynchronous, synchronous or isochronous, and atomic or non-atomic.

ODP objects encapsulate details of their implementation and provide an abstract view of their behaviour. The granularity of objects is not fixed by the reference model. The object concept can as well be used for modelling a whole information system in enterprise viewpoint specification, as for modelling a small protocol object in an implementation specification.

The object behaviour is specified as a set of interfaces. An interface represents objects role as a provider or exploiter of a service. As an object can support multiple interfaces, it can also participate in the provision of multiple services. A typical object supports at least a mission specific interface and a management interface.

An interface at which two or more objects meet for communication is not specified as an indivisible, global abstraction. Instead, both a client requesting a service and a server providing such a service, can have slightly different technical views of the interface. The ODP communication model concepts declare how these views can be mapped together in the binding process that creates a communication channel between the client and server interfaces.

The ODP model allows objects to appear in the system in two ways, by instantiation or by introduction. Instantiation is the often used model, where a template exists with sufficient information for the creation of an object instance. Introduction allows manipulation of objects without knowledge of their instantiation methods – only the object type is interesting. The ODP model defines type as a predicate that classifies objects based on their properties. A template is defined as a type detailed enough for instantiation. The instantiation process is naturally dependent on the platform facilities, and therefore, whether a type is also a template depends on the platform.

Object interfaces are bound based on their type, the object template is not considered. Because of the separation of client and server interfaces, also the sub-typing and substitutability concepts for ODP objects differs from other object models. The type system of objects focus on the shared features required from interfaces that need to be bound together, instead of focusing on implementation inheritance hierarchies. The essence of sub-typing rules for ODP objects is contravariance [11, 2]: the offered information flows must include at least the information expected by the receiver. In most object models, sub-typing is based on covariance: the replacing object can both expect to receive and offer more information that the replaced object expects and offers.

3 ODP structuring concepts

The ODP reference model introduces the structuring concepts of community, domain, and federation. These concepts can be used for organising objects for producing and exploiting services. The structuring concepts can be considered to be either static, design time concepts, or dynamic, operation time concepts.

A community is a configuration of objects with a common objective. For example, a pair of protocol objects form a community where the shared objective is to transport information from a computer to another.

A <X> federation is community of <X> domains. A <X> domain is a set of objects, each of which is related by a characterising relationship <X> to a controlling object. For example, a technology domain is the set of objects conforming to a technical standard, and a trading domain is the set of objects known to a trader object. An example of a federation is a trading federation, where two trader objects work together to serve the trading domains controlled by either trader.

4 ODP approach to conformance

The ODP reference model aims for specifications that allow software portability and interoperability. In order to guarantee portability, the implemented systems should conform to a strict standard specification. On other hand, the interoperability requirements force the specifications to allow heterogeneity of the system even at very high abstraction level. In order to accommodate both of these requirements, the ODP reference model defines four classes of reference points – points where the conformance to the standard specification need and is allowed to be tested. Other behaviour than that visible at these reference points is not considered.

The four classes of reference points are:

- A programmatic reference point specifies the limits for an object behaviour. Object implementations that pass the conformance test at this reference point can be replaced by each other.
- At a perceptual reference point there is some interaction between the system and the physical world.
- At an interworking reference point sets conformance requirements for communication between two or more systems in terms of the exchange of information.
- An interchange reference point defines conformance requirements for the manipulation of information at a physical medium. The manipulation rules are described in terms of access methods and formats. The goal is to ensure that physical storages can be transferred, directly or indirectly, to another system.

These structuring concepts are applied for the specification of a functionality of a system, not a whole system. Specifications of several interrelated functionalities can be allocated to same implementation objects. The development of ODP component standards give a good example of this structuring rule: trading function and type repository function are specified in different standards, although the same implementation objects can well support interfaces for both functions.

5 ODP transparent interaction model

The object can offer services through several interfaces that describe the actions in which the object can be involved. An interface is either an operation interface or a stream interface. An operational interface involves a set of operations, interrogations and announcements. A stream interface describes a set of continuous data flows, like audio or video flows. The activity associated to the stream interface is supported by a continuous data transfer protocol. Interrogations are allowed to have multiple terminations. This allows legal terminations and exceptions to be handled in a similar way. A termination may be exceptional but still the results of the operation are usable. Moreover, in a heterogeneous environment, the interfaces may be constructed to accept multiple correct terminations with equal status.

The binding and communication models aim for high-level communication primitives for programmers. For the management of heterogeneity and distribution some new concepts need to be available for the programmer, together with automatic tools for concept manipulation. A central concept is that of distribution transparency: programmers should be able to select a set of distribution transparency services for each communication primitive. The transparency services are implemented by the platform.

Distribution transparency is defined by RM-ODP [12] as follows: "The property of hiding from a particular user the potential behaviour of some parts of a distributed system."

Distribution transparency is selective in ODP systems [11]. The RM-ODP describes the following distribution transparencies:

- access transparency – masks differences in data representation and invocation mechanisms to enable interworking between objects;
- failure transparency – masks, from an object, the failure and possible recovery of other objects or itself, to enable fault tolerance;
- location transparency – masks the use of information about location in space when identifying and binding to interfaces;
- migration transparency – masks, from an object, the ability of a system to change the location of that object;
- persistence transparency – masks, from an object, the deactivation and reactivation of other objects (or itself);
- relocation transparency – masks relocation of an interface from other interfaces bound to it;
- replication transparency – masks the use of a group mutually behaviourally compatible objects to support an interface;
- transaction transparency – masks coordination of activities among a configuration of objects, to achieve consistency.

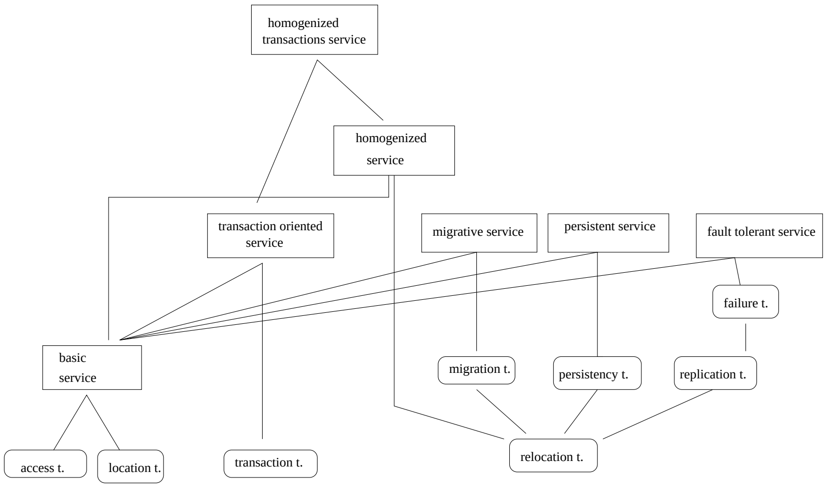

Figure 1: Hierarchy model of transparency concepts.

The ODP reference model does not yet define how these transparencies should be implemented. However, the reference model describes a hierarchy of transparency concepts so that a group of transparencies can be implemented together. The users can not select transparent services individually, but groups of transparencies. The hierarchy is illustrated in Figure 1. The implementation techniques of the transparency concepts include channel stubs, indirectness of references, and binding based of types instead of interface identifiers.

The distribution transparency aspects listed above are all selective for programmers. For instance, an interrogation can be defined to require migration transparent communication service from the platform. Therefore, the wide repertuary of selective transparencies separate ODP operations from the RPC style communication primitives.

In addition to the selective transparencies, the ODP reference model induces an additional transparency service, federation transparency, that must be embedded to the platform services. Federation transparency hides the problems caused by sovereign administration of the systems that support the communicating partners. Such problems include independent evolution of services, interface types, and information representation formats and languages. This capability differentiates ODP systems from the rest of open platforms [18].

6 Viewpoints

The ODP viewpoints allow object systems to be specified in an organised, guided manner. However, the viewpoint rules do not instruct on the level of detail or completeness of the specifications. Those aspects must arise from the software engineering process in which the viewpoint specifications are involved in. The relationship between the viewpoints and the software engineering process phases have been widely discussed and various interpretations have arisen. Officially, the ODP reference model does not bind the viewpoints to any specific software engineering process.

The ODP reference model defines five viewpoints – enterprise, information, computational, engineering and technology viewpoints [10, 11]. The viewpoint specifications of a system can be regarded as separate projections of a full system description. A system must be specified from each of the viewpoints. Each viewpoint specification is a consistent and complete specification on its own, but it only considers those aspects of the system that are valid on its point of view. So the viewpoint specifications do not overlap totally, but they may show different level of detail in the areas where they need to discuss same or related features. The engineering viewpoint specifications are tightly related with the ODP infrastructure model that is specified as part of the engineering viewpoint specification rules.

The enterprise viewpoint description of a system specifies the activities and the responsibilities of the system. Activity means any information exchange sequence and it is a high-level abstraction of the operations within the system. The system itself can have any granularity that is interesting. The system can be as wide as a global information network with all applications or as small as a memory cache in a processor. The enterprise specification identifies the system, its environment, and the required communication of the system and its environment. The specification answers to the questions "What is the purpose of the system?" and "What services the system is responsible to provide?" and "Who needs the services?".

The information viewpoint description of a system identifies logical information entities, their logical contents, their repositories and the objects that are responsible of the information flow in the systems. Questions for information viewpoint specification are "What information is needed to support the system's services?", "Where does the information come from and go to?", and "Is it necessary to store the information somewhere?". The information viewpoint specifications should not describe data structures, but only the semantics of the information. Also, the technique of storing information is irrelevant in this viewpoint (as the logical infrastructure supports storage services).

The computational viewpoint specification captures the behaviour of the system. Behaviour is an abstraction of how things are done, in contrast to the notion of what things are characteristic in enterprise viewpoint activities. An activity identified in enterprise viewpoint may involve several objects to perform a sequence of operations in computational viewpoint. The computational viewpoint shows the system as a composition of logical objects. For each object its interfaces are described. If the interface involves operations, each operation gets logical parameter descriptions (information structures, not data structures) – if the interface involves streams, each data flow component of a stream gets logical protocol descriptions instead. This is the viewpoint that usually explicitly shows potential for distribution. Neither the enterprise viewpoint nor the information viewpoint specifications need to express any distribution concerns. The computational viewpoint answers to questions like "Which operations are available?", and "Who (which logical entity) performs the operation?".

The engineering viewpoint specification identifies the infrastructure services needed for the system to operate. The ODP-RM engineering viewpoint defines the set of available infrastructure services, and all other engineering viewpoint specifications should show how the specified system utilise these services. The engineering specification therefore answers the question "By which services are the computational objects supported?". The ODP infrastructure model identifies a set of global, distributed basic services that should be available at each node in the global system. These include invocation of operations, transfer of continuous data as streams, trading, type repository functions, etc. These services facilitate selective transparency of communication between objects.

The technology viewpoint specification shows in a concrete hardware and software configuration how the system services and other required components are realized. The specification answers the question "How are the infrastructure services realized?".

The system specification includes five complete specifications, that all can be analysed as separate. Each viewpoint reveals a different aspect of the system, and therefore the full functionality can only be seen by looking at all specifications together. As the viewpoint specifications are all complete alone, the abstraction levels of objects in the specifications can differ. Still, the specifier must show how the specifications are mapped together. The usage of viewpoints helps the specification of large systems by separating concerns to separate specifications.

The ODP reference model introduces a vocabulary and a set of languages to discuss (distributed) systems, but it does not prescribe any special techniques to do this. Any specification language or technique that supports the same concepts can be utilised for ODP-style specifications. The problem with current formalisms that are close to the ODP concepts is that they do not support the level of dynamicity required in ODP specifications. However, development of tools and languages supporting the reference model would be very important and beneficial. When suitable tools are available, an ODP development technique could be formulated.

7 Engineering platform

The ODP reference model defines an abstract computing platform that comprises of a set of coordination, management, and repository functions. Each open system should independently support these functions, and in some restricted cases also allow other systems make controlled queries on the supported interfaces. The functions are described using a set of supporting concepts (nodes, capsules and clusters), that give an internal engineering view of the middleware software. However, these concepts are only used for descriptive purposes, to ease discussion, not as technology guidelines. In the following, we briefly view those functions that are most important for ODP conformant systems to participate.

Trading

Trading presents a global information repository. The global repository can be updated by independent information producers throughout the world-wide network, and the information users can create effective and large information searches. Trading is not a mapping mechanism like name services. Instead, it resembles more directory services that allow attribute values to be used as a search criteria.

Trading activities are described through a trading community that represent the roles of 'importer', 'exporter' and 'trader' [13]. The object in trader role supports a repository of 'offers'. Each offer describes properties of an entity. The offers are produced by objects in exporter roles. When an exporter sends an offer for a trader to be stored, it is said to 'export'. When an exporter requests an offer to be deleted, it is said to 'withdraw' an offer. The objects in importer roles make queries to the offer repository, they 'import'. The import requests have a basic form that is similar to database queries: the request specifies criteria for selecting the offers to be included to the response. The objects that use the traded information need not necessarily be the importers or exporters themselves. The 'clients' and 'servers' that use the information are therefore considered as separate roles. The communication between importers and clients, or between servers and exporters is not prescribed (but trading mechanism is recommended).

The ODP trading function is designed to mediate server interface offers. An interface offer contains an abstract service type name, an interface signature, and a set of attributes. The attributes can describe quality of service aspects or they can describe the supporting platform features of interest. The trading function specifies only the mediation mechanism and information structuring rules, not the interpretation of offers.

Type repository

The type repository services [14] mediate type information that is available at runtime. Traders use type repositories for checking conformance between object interfaces. Traders could include the type management facilities themselves, but separation of these tasks to an independent module provides flexibility of configuration and supports independent evolution of the type systems.

The type repository supports operations for

- publishing type descriptions,
- checking conformance of two type descriptions,
- retrieving subtypes and super-types,
- translating types, and
- name management operations for types.

Bindings

The engineering model of ODP also defines the obligations of channels between object interfaces. Most important contribution is that ODP model supports multi-party communication, in contrast to, for example, OSI or OMA models. In addition, the federation transparency and the selective transparencies must be supported (to the degree claimed – each system can claim a given conformance level that requires a collection of transparency services). A valuable addition to the communication model is management of quality of service contracts [6, 15]. This aspects is essential for the development of modern multi-media applications.

8 Conclusions

The family of ODP standards first defines a consistent and a rigorous set of concepts and the related terminology for analysing, designing and standardising open and distributed systems. In addition, a set of viewpoints is specified for isolating features of a system for analysis in a consistent manner. Second, a modern architecture guideline is specified for designing computing platforms. This guideline, i.e. framework, identifies a set of common functions necessary for managing system evolution and cooperation across autonomous administrative domains. In the family of standards, these functions are further standardised.

The ODP object model differs considerably from other object models. Instead of defining a object-based methodology, it only offers a set of concepts. The object related concepts are more rigorous than for example the object model of OMA or TINA, or some programming languages. First of all, it introduces concepts for stream interfaces. Stream facilities are missing for example from OMA model. Moreover, the ODP object model elaborates the interface concepts: In many object models, interfaces are characterised as message exchange taking place when an operation is invoked at a server object. The data formats of the exchanged messages are fixed. In ODP model, interfaces are defined via communication of information and each participant of the communication explicates the assumed structure of the interface.

Most object-oriented programming languages interfaces are defined only as a set of operation signatures, not considering behaviour. Semantical conformance, that is required for ODP type binding of objects, can be assured only though inheritance of interfaces. However, in a federated environment, the sovereign systems can not facilitate such a shared inheritance hierarchy.

The ODP reference model specifies an open platform with middleware services like trading and type repositories. The OMG/CORBA platform, that has gained a defacto standard position as the ODP platform implementation, does not yet include all required services.OMG has already adopted the ODP trading function for the OMG object trader service [20]. There is also an existing project on Meta-Object Facility [21], that is closely related to ODP type repository function. However, as the ODP type repository is focused on services required by the open platform at run-time, the MOF service is focused on services for programming tool production and design.

ODP viewpoints are not layers, but projections of the specified system. Viewpoint languages are meta-languages that indicate the concepts of concern, without an enforced notation. The selection of an object-based notation often creates confusion in the specification reader, because the notation is always designed for a contradictory object model. Therefore, an ODP specific set of languages would be welcome.

Viewpoints are neither considered as part of a specific software engineering process, although such mapping is often assumed. Specialisations of the ODP framework, like TINA, use their private conventions for mapping viewpoints to software engineering processes. None of the mappings should be expected to suite all application areas equally.

References

[1] Blair, G., and Stefani, J.-B. *Open Distributed Processing and Multimedia.* Addison-Wesley Publishing Company, 1997.

[2] Castagna, G. Covariance and contravariance: conflict without cause. *ACM Transactions on Programming Languages and Systems 17*, 3 (1995), 431 – 447.

[3] Chapman, M., and Montesi, S. *Overall concepts and principles of TINA.* Telecommunications Information Networking Architecture Consoritum (TINA-C), Feb. 1995. Also http://www.tinac.com/95/overall/public/Overall/overall.ps.

[4] Group, I. O. E. *NATO Open system environment: glossary of terms*, June 1996. Version 3. Available at http://www.nacisa.nato.int/NOSE/GLOSS1.HTM.

[5] ISO/IEC. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – ODP Naming framework*, Jan. 1997. DIS14771.

[6] ISO/IEC. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – ODP Interface References and Binding*, July 1997. FCD14753.

[7] ISO/IEC. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – Protocol Support for Computational Interactions*, July 1997. ISO/IEC/JTC 1/SC 21 7N1313.

[8] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection – Systems management – Open Distributed Management Architecture*, 1996. DIS13244.

[9] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing.*, 1996. IS10746.

[10] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 1: Overview*, 1996. IS10746-1.

[11] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 3: Architecture*, 1996. IS10746-3.

[12] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 2: Foundations*, 1996. IS10746-2.

[13] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. ODP Trading function. Part 1: Specification*, 1997. IS13235-1.

[14] ISO/IEC JTC1. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. ODP Type repository function*, 1997. CD14746.

[15] ISO/IEC JTC1/SC21/WG7. *Working document on QoS in ODP*, Jan. 1997. N1192.

[16] ITU-T. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing.*, 1996. Recommendation X.902.

[17] ITU-T. *Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. ODP Trading function. Part 1: Specification*, 1996. Recommendation X.950.

[18] Kutvonen, L. Why CORBA systems cannot federate? In *OMA/ODP Workshop* (Cambridge, UK, Nov. 1997).

[19] Linington, P. RM-ODP: The architecture. In *The 3rd International Conference on Open Distributed Processing – Experiences with distributed environments* (Brisbane, Australia, 1995), K. Raymond and L. Armstrong, Eds., Chapmann & Hall, pp. 15–33.

[20] Object Management Group. *OMG RFP5 Submission: Trading Object Service*, May 1996. Also http://www.omg.org/docs/orbos/96-05-06.ps.

[21] Object Management Group. *Common Facilities RFP-5: Meta-Object Facility*, 1997. OMG TC Document cf/96-05-02.

[22] Object Management Group and X/Open. *The Common Object Request Broker: Architecture and Specification*, Nov. 1993. Also http://www.omg.org/docs/1991/91-12-01.ps.

[23] Open Software Foundation. *OSF DCE User's Guide and Reference*, 1994.

[24] Raymond, K. Reference model of open distributed processing (RM-ODP): Introduction. In *The 3rd International Conference on Open Distributed Processing – Experiences with distributed environments* (Brisbane, Australia, 1995), K. Raymond and L. Armstrong, Eds., Chapmann & Hall, pp. 3–14.

Biography

The author is the Finnish National Body delegate in ISO/IEC JTC1/SC21 WG7 that progresses ODP standards. At the moment, SC21 is closing and the ODP work is moved under SC33.