

# Management of application federations

*Lea Kutvonen*

*Department of Computer Science, University of Helsinki*

*PO Box 26, FIN-00014 University of Helsinki, FINLAND;*

*Lea.Kutvonen@cs.Helsinki.FI*

## **Abstract**

Open system architectures, like RM-ODP, encourage integration of applications by dynamic federations of sovereign application objects. A federated application architecture can be based on three cornerstones: (i) the separation of application configuration from communication and computation, (ii) the exchange of meta-information about applications, computation resources, and communication services between software components, and (iii) the disjoint operation of applications at autonomous administrative domains. The federated application architecture supports flexible interoperation between heterogeneous computing environments by allowing interoperation relationships to be formed automatically. In this paper we describe how run-time application management can be a local activity at each domain, without a need for checking global consistency. The mechanism is based on dynamic contracts (e.g., agreed quality of service) that are managed as properties of the interactions between independently administered application objects.

## **Keywords**

Open distributed processing (ODP), software architecture, application management, interoperability of heterogeneous systems

## 1 INTRODUCTION

Integrated services are tenaciously demanded on areas where information services form an essential part – such as international banking services, or teleconferencing. The modern applications are integrated from a set of autonomously produced services. For example, a teleconferencing service exploits audio and video recording and replay services independently at each conference loca-

tion, and several underlying data transport services between the locations. The data transport service may in turn utilise external billing services.

Open system architectures, like RM-ODP (Reference Model of Open Distributed Processing) (ISO 1996*b*, ISO 1996*c*) and TINA (Telecommunication Information Network Architecture) (Chapman & Montesi 1995), encourage applications to be distributed across autonomous systems. Even independent applications can be dynamically integrated via federations. In this paper we build a management model for federated applications. We build on previous work on configuration languages and quality of service (QoS) negotiation and apply those ideas in the context of a distributed application architecture.

We build our overall model, a ‘federated application architecture’, on three cornerstones: (i) the separation of the application configuration from the communication and computation tasks in the application, (ii) the exchange of meta-information – like service type and QoS – about applications, and about computation and communication services, and (iii) disjoint operation of applications at autonomous administrative domains. The objects considered in this paper are composed objects, like packages in the TINA architecture. Each object is an independent application that is administered on a single administrative domain, but it can access services from other administrative domains. An application federation occurs at operation time, when a service is accessed across administrative domain boundaries. In other words, a ‘federation’ is an interoperation relationship where each participant is under separate management. This is in contrast to the strive for global management functionality in many distributed system architectures.

Our management approach for federations of applications allows the interoperation relationships to be formed automatically, thus decreasing system maintenance cost in comparison to distributed architectures. In addition, the interoperation relationships can be formed so that they govern the actual co-operators only, and do not restrict other potential cooperation relationships for the system. Currently, application program code often need be modified for each cooperation relationship, because of the global management scheme adopted by most distributed systems. It is also common that the cooperation relationships are realized by joining the administrative domains, which unnecessarily restricts the autonomy of the systems.

In the federated architecture, we identify a set of local management points and describe what methods can be applied for federated management using these management points. The approach differs from traditional distributed management approach by denying direct control of remote objects. Instead, a multiway agent – a contract object – is created to monitor, negotiate, and instruct management operations. This approach does not require that the cooperating systems fully work in a consistent manner, but allows a small, consistent domain to be created for the duration of the interaction.

The paper is organised as follows. Chapter 2 reviews the facilities provided by the ODP architecture, and Chapter 3 reviews distributed management aspects. These chapters set the vocabulary and the scope for the discussion.

Chapter 4 introduces the federated application architecture, contracts, and management points, while Chapter 5 discusses federated management methods. These chapters are interlinked: Chapter 4 shows the location and the semantics of the information, Chapter 5 shows the mechanisms used for manipulating that information. Chapter 6 discusses the effects of our approach to the ODP architecture.

## 2 OPEN SYSTEMS AND FEDERATIONS

### 2.1 Openness and interoperation

Traditionally, openness has meant the public availability of interface descriptions. However, in modern systems this is not sufficient any more. The public availability supports only interoperation establishment in the design phase, whereas the current requirement is to allow the same freedom at operation time (ISO 1997a). The requirements for interoperation include interconnection between the computing systems, but also shared understanding of the following aspects: the interpretations of information, the semantics of the computations, and the access technology of the service. The autonomy of the organisations providing the services, however, creates a major problem. The organisations have no shared control and can therefore independently select the computation and communication technologies suitable for their private needs. The application services they provide may also evolve independently from each other, thus providing a possibility to modify the semantics of information and computations. In this kind of environment, interoperation can be established at operation-time only by dynamic exchange of meta-information about software components. We will further discuss meta-information in Chapter 2.3.

The interoperation of services occurs on three dimensions that we can illustrate by a teleconferencing example:

#### **(i) Interoperability within or across application domains**

We can organise a plane of application domains as layers of an abstract, hierarchical computing system. The layers start at the bottom from hardware, rising via operating system and communication system services to middleware services and finally reaching end-user level applications like the accounting services, and the audio and video services of the teleconferencing application. The middle layers in this hierarchy need to establish interworking relationships between computing systems at different locations. In an 'interworking relationship' all communicating parties are explicitly aware of the protocols, data representations, and other necessary details. The upper layers of services can be supported by lower layers in such a way that the semantics of information and computation can be designed and evolve locally (e.g., within an organisation) without a global agreement. In an 'interoperation relationship' the communicating parties are not obliged to be explicitly aware of the transport and representation details. The lower layers can be exploited when

building shared understanding at the middleware level in order to provide a virtual platform for each of the application objects.

**(ii) Interoperation across administrative domains**

At each physical location there can be one or more independent administrative domains effective. For example, at a named computer, there may be independent control for data transport and for billing services. In our example, the teleconferencing packages at each administrative domain may be differently structured, i.e., one package may include a single object whereas the other separates audio and video.

**(iii) Interoperation across physically distributed locations**

The computation of a service can be performed at a set of physical locations. It is by no means necessary to introduce autonomy for each location. Instead, centrally controlled distributed services are encouraged by the architecture. For example, data transport services by nature must have a administrative domain that covers multiple computers.

## 2.2 Federations

A ‘federation’ (ISO 1996*c*, cl. 5.1.2) is an interoperation relationship between objects at the same application domain, but under independent control. The object’s autonomy may have various reasons: the federation may be established due to differences in processing technology, language, life-cycle management, or policy. The federation requires an interworking relationship to exist at lower layers of the application plane.

Federation is often considered to be a contractual relationship between organisations or the computing systems of the organisations (Beasley et al. 1994). In contrast to that, we concentrate in this paper on the federations of applications. The aspects of system federation are considered to cover compatibility of resource administration, accounting, interworking mechanisms, and application-specific features like interfaces and QoS (Beasley et al. 1994). Because we only consider application federations, we need to consider only the application-specific features and the cooperation of the application with the infrastructure that supports all necessary interworking relationships, including accounting and resource control.

Federations between independent application objects require the support of open, federated platforms for distribution transparent access to services. In addition, the application objects need to support locally the full application service, and need to be able to request similar service from other similar application objects remotely. For instance, the teleconferencing application objects should be able to record and replay activities at the conference location in a similar way for local and remote requests, and to be able to request remote recording and replay.

Open computing platforms support dynamic repositories for meta-information about the application and platform interfaces (e.g., ODP trading function (ISO 1997*b*), ODP type repository function (ISO 1997*c*), CORBA interface repository (OMG 1996), and CORBA meta-object facility (OMG 1997)).

### 2.3 Meta-information in RM-ODP

The meta-information concepts of the RM-ODP include type (ISO 1996*b*, cl. 9.7), template (ISO 1996*b*, cl. 9.11), and contract (ISO 1996*b*, cl. 11.2.1); the meta-information publication mechanisms include trading (ISO 1997*b*), type repositories (ISO 1997*c*), and naming functions (ISO 1997*d*). Also OMG is designing a meta-object facility (OMG 1997), but this service is aimed for the construction of distributed application design tools.

The ODP model is based on objects communicating via interfaces. The objects become available either by ‘instantiation’ (ISO 1996*b*, cl. 9.13) from a template or by ‘introduction’ (ISO 1996*b*, cl. 9.16) with a type. The difference between concepts of template and type is subtle. A ‘type’ is a predicate that allows reasoning about the properties and the behaviour of an object. A ‘template’ is a type that is detailed enough to be used for instantiation at a given platform. In a heterogeneous environment a predicate expression thus can be seen as a template in some subsystems and in some others not.

When objects need to communicate, their interfaces must be bound together. The details of the ODP binding process have not yet been fully standardised, but a general framework already exists (ISO 1997*a*). The binding process essentially collects information about the capabilities of the client and server role interfaces and establishes a ‘contract’ that can be fulfilled by all involved interfaces. The contract creation process should ensure that the interfaces exist or can be instantiated, the interfaces are compatible, and that there is no (e.g., security related) prohibition for the binding. The compatibility requirements for interfaces capture conformance of interface signature types, conformance of interface behaviour, and availability of a shared data transport mechanism. The requirements also capture shared understanding of the names used for types, behaviours, and protocols in the binding process. Furthermore, an agreement on QoS is included. What attributes are suitable for measuring the QoS depends on the service in question, e.g., availability, throughput, jitter, failure recovery protocol (ISO 1997*e*).

Each party in the cooperation relationship need, for the purposes of the life-cycle services and the binding process,

- information about the available object interfaces and the properties of those interfaces; and
- a shared, mutually consistent view to available interface types.

The trading function can be used to mediate information about interfaces. It is a global, federated mechanism that provides the means to advertise and to discover a particular service type (ISO 1997*b*). The trading community includes ‘importers’, a ‘trader’, and ‘exporters’. The client is an application object that wishes to find a server. The server represents a remote application object of interest. Neither of these objects need to be involved in the trading action themselves, but they have representatives, importers and exporters. The trader stores information about available servers as ‘service offers’. Service

offers are grouped so that each set of service offers represents providers of one abstract service type. Each service offer includes an interface reference that conveys where and how the service can be invoked, and a set of property values that convey other aspects of the service, such as QoS. The properties can be either static (e.g., processor type) or re-evaluable at usage-time (e.g., queue length). The service offers can also convey policy choices of the objects, e.g., what kind of search algorithm is used in a database.

The type repository function can be used to mediate the type information and to translate it to template information as needed. The type repository function (ISO 1997*c*) resembles the trading function by its behaviour, but instead of interface instances it mediates type information. The key concept for type repository is 'type description'. A type description expresses an abstract service class and gives a set of concrete expressions in different languages for it, i.e., the type description contains a set of 'type definitions'. The different expressions allow replacing one type definition by another. This is especially important when a declarative type expression can be related to a template. A supplementary concept for type repository is 'type relationship' that is used for expressing conformance between type definitions administered separately. For the federation of the systems the relationship is essential: conformant type descriptions are interchangeable (or interceptable).

Both the trading function and the type repository function trust on a federated naming service. The ODP naming framework standard under development (ISO 1997*d*) defines a federation mechanism between global naming systems with independent management. However, it still requires globally distributed management of a single naming system and does not exploit federation facilities on this aspect (Kahkipuro et al. 1997).

### 3 ASPECTS OF MANAGEMENT

The term 'distributed management' can refer both to central management of distributed objects, and to distributed control of some objects. Management of distributed object systems (e.g., ISO 1992) allows full control of the target system from a single administration point. Distributed management applications (e.g., de la Fuente & Walles 1994) allow several administrators to manage the same distributed system under the control of a shared tool. The tools actually control that the administrative operations are not contradictory to each other, thus making the tool to control the administrators in addition to the target system.

Distributed management can be applied to applications, systems, and communication. Examples of managed applications include the aforementioned teleconferencing systems and banking services; examples of managed computing systems include distributed computing platforms with file services, processing capacity, storage capacity, etc. (OMG 1996); examples of managed communication systems include OSI data transport services and TCP/IP ser-

vices (ISO 1992, Warriier et al. 1990, OMG 1996, ISO 1996*a*, de la Fuente & Walles 1994).

The management functions relevant for these areas of open systems include configuration management, QoS management, monitoring, and policy management (ISO 1992, Sloman 1990). The configuration management specifies which software objects are needed for the application to perform its service. The configuration management includes software version control, specification of object classes and instances, and the binding of interfaces and software object allocation to independent platforms.

The QoS management involves specification of required and offered QoS, negotiation process to agree a QoS level, maintaining the agreed QoS, notification of changes in QoS, and renegotiation of the modified QoS agreement under changed circumstances (ISO 1997*e*). The QoS management is a generalisation of performance and fault management of OSI management functions (ISO 1992). The QoS management requires monitoring of state, errors, performance, and usage information. The QoS change notifications can only be produced via monitoring the target object behaviour.

The policy management involves specification of a policy and the provision of the policy as part of the target object behaviour. Policies can be assigned jointly for domains of objects. For instance, a storage policy of an application group may state that a certain file name must be used for passing data between the applications within the same group.

Each of the aforementioned aspect could also be considered at design or implementation time. However, the benefits for the federated architecture can be gained mainly through the run-time management functions. Therefore, we concentrate on run-time management aspects of applications.

## 4 MANAGEMENT POINTS

In order to introduce the federated management architecture, we must identify the location and the semantics of the management information, as well as the management methods that manipulate the information. In this chapter we concentrate on the information itself, using teleconferencing as an example; in Chapter 5 we discuss the methods. We remind that an application is considered to be controlled within a single administrative domain. Therefore, the management aspects can always be applied locally.

The first kind of management points is the instantiation of sovereign applications, i.e., object factories. The factory is local for each domain. In our example, there must be a self-contained teleconferencing application for the users at each location. The applications should be independent and interoperable, instead of being components of a single application. The separation of the applications allows independent implementations and independent management at each domain. Chapter 5 describes how a federation of application objects can be instantiated although the participating domains may use different methods for expressing and establishing (configurations of) objects.

The second kind of management points is the management of communication contracts, either at the application level or at the data transport level. The contracts collect details of communication relationships: protocols, QoS agreements, addresses of participants, etc. The contracts must in some cases be explicit objects, because they must support explicit contract modification operations. We remind that the scope of a contract, or a federation, is restricted to the interoperation between objects, and does not govern the administrative domains in large. We study the contract management further in Chapter 5. In the following paragraphs we study the nature of two contracts in our example system: one at the application level, second at the data transport level.

The teleconferencing applications are virtually bound together via stream interfaces that allow flows of audio and video to be exchanged. For the interoperation to succeed the interfaces need to be implemented so that the following aspects of cooperation are fulfilled:

- each application receives enough resources and remote services to be able to produce high enough quality of service for its users; the QoS attributes in the example case can express the use of colours and frame frequency in the video flow, and synchronisation between the video and audio flows;
- agreement on a protocol to change QoS agreements;
- agreement on a conference protocol that defines what operations are available, for example, joining or withdrawing a site to the conference; and
- agreement on how to behave in situations where an application or the virtual platform fails to behave according to the agreement.

These aspects are collected in to a communication contract (ISO 1997*a*). This communication contract should not be mixed with the environment contract (ISO 1996*b*, cl. 11.2.3) related to the object, although also the environment contract involves QoS aspects (Leydekkers & Gay 1996). The environment contract governs the object behaviour under some circumstances created by the local platform services. The communication contract covers only part of the environment contract.

Referring back to the application plane (Chapter 2) we see that each of the telecommunication applications must interoperate with the communication network services in order to realise the virtual communication relationship at the application level. It is reasonably easy to establish these two bindings, one for each of the conferencing services with their local communication interfaces.

The communication service is at a different level in the hierarchical computing system and has different administrative domain borders as the teleconferencing service layer. The end-points of the communication lines are at the same physically distributed domain and this domain has a distributed administration. The control of the communication requires a strict interworking relationship among all physical locations involved. If any differences in the communication services exist, the communication management tools must do explicit transformations or the communication fails.

For the teleconferencing application and the communication network to interoperate the following agreements are needed:

- QoS, such as channel throughput and jitter;
- event notification if the channel fails to transport data or fails to meet the QoS agreed;
- transport protocol and transport channel endpoints; and
- interface for QoS contract modifications.

Also these aspects are collected in to the communication contract (ISO 1997*a*).

## 5 CONTRACT MANAGEMENT

We have now reduced the management of federated applications to the management of contracts. In this chapter, we study the nature of contracts and the management activities they support. Then, we apply contracts to object instantiation and binding management across autonomous domains.

A contract is an information object that collects details of a communication relationship. The contract structure depends on the application domain and the contract itself is negotiated between two or more application instances. Technically, the contract information is replicated for each of the applications in the federation. Because the interfaces can reside at separate domains, the replicas may have different data representation formats and coding. The replicas of the contract have a protocol among themselves to keep the contract information consistent. Each application can use that local contract replica as policy information or as parameter for its internal activities. This mechanism can be used to provisioning of the contract as part of the object behaviour (an example in Meyer & Popien 1995).

A contract is a distributed engineering object and it offers a management interface with modification operations for objects within the federation. However, the contract must obtain a permission for each change from the applications in the federation before it can commit a requested change. Therefore, each application object must have interfaces that allow the contract object to enquire their capabilities and policies. A modification operation is rejected if a local policy does not allow the requested modification. In this way, each application is able to protect its local policies, and simultaneously the federated behaviour stays consistent.

All involved applications do not need to be able to manage a dynamic contract, i.e. to be able to response to requests of contract changes. In those cases, change requests are denied. A static contract can be formed by matching together the offered and the required features of each involved interface, and creating an object to capture the matching result.

The contract management can be applied to object instantiation and application configuration. Applications are traditionally expressed as templates (programs) for the platform in use. However, in the open architecture, we base the instantiation process on descriptive configuration information, i.e., type information, instead of directly expressed templates. Instead of a full template for the target platform, the application can be expressed as a configuration of services with a certain type and quality. An example of such

a programming style can be found in Darwin language developed in Regis project (Magee et al. 1994). To the type-base descriptive specification of the application we can integrate QoS relationship, as it is possible to do in TINA ODL (Parhar 1996). This gives only place-holders for the QoS contracts. When the instantiation takes place, the environment contract for the application is defined and from that information all the QoS requirements can be induced for the forthcoming communication contracts and platform services.

The platform on which such descriptive applications can be run requires a conveniently large set of object implementations and a repository of relationships between these object implementations and types. The ODP trading function serves this mapping task, and also the CORBA implementation repository (OMG 1996). When the platform is heterogeneous it is essential that it also includes a type repository that reveals different expressions of the same type. As we mentioned earlier, a type description may be a valid template in some other run-time environment. For different run-time environments the same type needs to be instantiated differently. The ODP type repository function is a suitable repository for relating abstract types and concrete templates in heterogeneous environments.

The actual instantiation process first considers the type descriptions (like contracts) and selects a suitable type specification for the local platform, and then selects a suitable server for that type. These steps can be supported by the type repository and the trader. After that the instantiation process must consult the platform management functions, i.e., node management, cluster management, capsule management and object management, as described in the architecture descriptions of ODP and TINA (ISO 1996c, ISO 1996a, de la Fuente & Walles 1994).

The contract management concept can also be applied to bindings between objects. A binding can be established when all layers from the application to the hardware can establish a contract. In our teleconferencing example, the interoperation negotiation process must consider two levels of contracts. The teleconferencing interoperation contract restricts the types of services that are suitable at the lower layer. In the communication layer in turn, the requirements are used to select templates that suit the required type. The selection of the template is affected by the interoperation partner capabilities as well as the local platform capabilities. Therefore the capability information needs to be available, for example through traders and type repositories.

In the binding establishment process, the contract expresses an agreement on 'channel type' (ISO 1997a). The channel type can be realised as a type-based instantiation process, where each of the computing nodes selects a suitable template for the shared channel type.

When a binding is established, it must be monitored, and if a breach of its contract occurs, the binding may possibly be reconfigured. It should be noticed that the breach may be related either to the federated application or to the virtual platform supporting the application federation. In the latter

case, the application federation (compare to the liaison in (ISO 1997a)) can be retained and only the lower service layer need to be reconfigured. This means that the channel type is retained, but the instances for it are replaced.

The general view of the above described management architecture differs from trading-based approaches (Kovacs & Wirag 1994, Tschichholz et al. 1996, Beasley et al. 1994) in one aspect. In our approach, there are no general application management functions that control and monitor a federated application as a whole. The application itself causes independent management actions to occur at the autonomous domains. Trading decisions, that are performed on the request of the application, only induce management actions. However, the traders and type repositories are still used for federated discovery of services (i.e., not plain resources), like in (Ni & Goscinski 1994). The application management in a federated environment becomes a set of independent management activities, instead of a single, distributed management activity. At each domain the available objects can be independently created, modified, advertised and withdrawn. The federated application configuration exploits objects that are available at the time, only depending on their types.

## 6 EFFECTS TO RM-ODP

The contract-based management architecture suggests some changes to the RM-ODP computational model, the QoS work, and the binding framework.

It has been suggested (ISO 1997e) that the RM-ODP computational model should include resources as objects (in the example, the communication layer services). Instead, we think that the contracts should be included as computational binding properties. Modelling QoS as binding properties supports type-based instantiation, because the QoS contract can be used as part of the type specification. Modelling QoS as binding properties also allows the contracts to be modified separately, in addition to the direct management of each component object in the channel that realises the binding. This design has the benefit that the application has a single interface on which it can try to change the contract. A modification request initiates a renegotiation of the contract. If the renegotiation fails, no changes are done; if the renegotiation succeeds, it initiates the reconfiguration of the channel according to the new contract. In some cases the channel is not even instantiated at the time of contract modifications. For example, the channel may have a policy-driven resource reservation and releasing protocol that is based on traffic, i.e., the binding may not always have an existing channel.

The QoS in ODP work (ISO 1997e) defines the contract negotiation process with the terms of QoS relationship, QoS requirement, QoS offers, and QoS agreements. The QoS relationship defines the circumstances under which a certain QoS level can be offered. However, in a federated environment that circumstantial information can not be available and must not be manageable from remote domains. Within the local domain the circumstances can be monitored and probabilistic information can be collected and provided for

remote domains. With this modification the negotiation process is suitable for the federated binding process. The same negotiation process can be used for the rest of the contract aspects as well. We can collect all these aspects to a single offer, requirement, and agreement according to the QoS negotiation process. This negotiation process is often a matching problem and therefore it can be assigned to the trading function.

The current binding framework (ISO 1997a) allows only the management of channels, i.e., resource objects. This specification corresponds well to the suggestion that resource objects are explicitly introduced to the computational model. As we consider the contracts to be a more dominant concept and manage them as communication properties, the model requires an addition: the contract needs to be managed independently of the channel management. The current binding framework (ISO 1997a) already introduces a liaison as an information object, and it can be extended to support liaison control activities.

## 7 CONCLUSION

The application management approach described in this paper differs from traditional distributed management approaches by denying direct control of remote objects. This restriction is necessary because the applications can acquire services from independent administration domains. Between the administration domains there can not be any cross-control relationships. Instead, each domain is managed by a private controller and this controller makes all management related decisions. The federated management methods must feed these controllers with contract and monitor information in order to force them to make suitable management decisions. We also recommend, that the contracts are included as properties of the communication actions between computational objects. As a contract information exchange and negotiation method we can use the trading function that is available in all open systems. However, for each application area we need to standardise a contract structure. Otherwise, the controllers and traders do not have a common information base that is necessary for establishing federations.

The application life-cycle and interface binding mechanisms introduce negotiation activities and physically distributed, even heterogeneous decision making activities. For these interworking relationships the above referenced systems give excellent solutions.

In an open system environment in which federated, sovereign application objects need to be managed, we face several problems that include

1. expressing configuration information in a federated environment, where each participating system may use a different method of expressing and establishing configurations;
2. expressing and negotiating QoS contracts when the expression languages may differ;

3. maintaining an agreed QoS level at remote systems when there are no direct control mechanism to the monitored resources;
4. creating configurations of objects that follow independent policies and that have independent policy management causing asynchronous policy changes.

These problems can be solved by the open architecture. The ODP type repository function shows an example how to collect together different concrete expressions (heterogeneous set of templates) for the same abstraction (type). In each environment the most constructive expression can be used. Mechanisms like type repositories solve problems 1 and 2. The control of remote resources, problem 3, must be solved via contracts. The contract structure on a certain application area gives a firm basis for interpretation of required, offered, and agreed values. The configuration management services, like binding and instantiation, must follow the values given in the contracts. If this rule is followed, no direct control of resources at remote domains is needed. The mechanism ensures that a predictable behaviour is met also with independent systems. The contracts and trading of contract information also solve the fourth problem. When behaviour information is always available from traders, the system can use a server that follows a suitable policy. The policy management and normal services of an object must of course be implemented so that they do not interleave.

## REFERENCES

- Beasley, M., Cameron, J., Girling, G., Hoffner, Y., van der Linden, R. & Thomas, G. (1994), *Establishing co-operation in federated systems*. APM.1128.02.
- Chapman, M. & Montesi, S. (1995), *Overall concepts and principles of TINA*, Telecommunications Information Networking Architecture Consortium (TINA-C). Document TB-MDC.018-1.0-94.
- de la Fuente, L. A. & Walles, T. (1994), *TINA Management Architecture*, TINA-C. TB\_GN.010\_2.0\_94.
- ISO (1992), *Information Processing Systems – Open Systems Interconnection, Systems Management Overview*. IS10040 | X.701.
- ISO (1996a), *IT – Open Systems Interconnection – Systems management – Open Distributed Management Architecture*. DIS13244.
- ISO (1996b), *IT – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 2: Foundations*. IS10746-2.
- ISO (1996c), *IT – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. Part 3: Architecture*. IS10746-3.
- ISO (1997a), *IT – Open Systems Interconnection, Data Management and Open Distributed Processing – Interface References and Binding*. CD14753.
- ISO (1997b), *IT – Open Systems Interconnection, Data Management and*

- Open Distributed Processing. Reference Model of Open Distributed Processing. ODP Trading function. Part 1: Specification.* IS13235-1.
- ISO (1997c), *IT – Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. ODP Type repository function.* CD14746.
- ISO (1997d), *IT – Open Systems Interconnection, Data Management and Open Distributed Processing – ODP Naming framework.* DIS14771.
- ISO (1997e), *Working document on QoS in ODP.* N1192.
- Kahkipuro, P., Kutvonen, L. & Marttinen, L. (1997), Federated naming in an ODP environment, in ‘Joint international conference on Open Distributed Processing (ICODP) and Distributed Platforms (ICDP)’, pp. 314–325.
- Kovacs, E. & Wirag, S. (1994), Trading and distributed application management: An integrated approach, in ‘5th IFIP/IEEE Workshop on Distributed Systems: Operations & Management’.
- Leydekkers, P. & Gay, V. (1996), ODP view on quality of service for open distributed multimedia environments, in ‘4th Workshop on Quality of Service’.
- Magee, J., Dulay, N. & Kramer, J. (1994), ‘Regis: a constructive development environment for distributed programs’, *Distributed Systems Engineering Journal* 1(5), 305–312.
- Meyer, B. & Popien, C. (1995), Flexible management of ANSAware applications, in ‘3rd conference on Open Distributed Processing’, pp. 255–265.
- Ni, Y. & Goscinski, A. (1994), ‘Trader cooperation to enable object sharing among users of homogeneous systems’, *Computer Communications* 17(3).
- OMG (1996), *The Common Object Request Broker: Architecture and Specification.* OMG Document No. 91.12.1. (Revision 2.1.).
- OMG (1997), *Common Facilities RFP-5: Meta-Object Facility.* OMG TC Document cf/96-05-02.
- Parhar, A. (1996), *TINA object definition language manual. Version 2.3,* TINA-C. Document TR\_NM.002\_2.2\_96.
- Sloman, M. (1990), Management for open distributed processing, in ‘Future Trends of Distributed Computing Systems’, Egypt, pp. 533–539.
- Tschichholz, M., Tschammer, V. & Dittrich, A. (1996), ‘Integrated approach to open distributed management’, *Computer Communications* 19(1), 76–87.
- Warrier, U., Besaw, L., LaBarre, L. & Handspicker, B. (1990), *The common management information services and protocol over TCP/IP (CMOT),* Network Working Group. Request for Comments: 1185.

## BIOGRAPHY

Lea Kutvonen holds a Ph. Lic. degree in computer science from the University of Helsinki. She is a member of faculty at the department of computer science since 1990. Her interests include distributed systems, trading, and ODP architectures. Currently she works for the completion of her Ph.D. thesis. She is a member of ACM, IEEE, and Finnish Data Processing Association.