# Multimedia Services in Open Distributed Telecommunications Environments

Peter Leydekkers

CTIT Ph.D-thesis series No. 97-12

P.O. Box 217 - 7500 AE Enschede - The Netherlands
Telephone +31-53-4893779 / fax +31-53-4333815

**C**entre for
**T**elematics and
**I**nformation
**T**echnology

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Leydekkers, Peter

Multimedia Services in Open Distributed Telecommunications Environments /
Peter Leydekkers. -Groningen: KPN Research. -I11.
(CTIT PhD-thesis series, ISSN-1381-3617; no. 97-12)
Proefschrift Universiteit Twente, Enschede. - Met lit. opg., ref.
ISBN 90-72125-04-6
Trefw.: ODP-RM, multimedia, stream interfaces, synchronisation, multimedia conferencing systems,TINA, open distributed systems, quality of service.

Cover picture inspired by ‘La Dance’ from Henri Matisse.

# MULTIMEDIA SERVICES
# IN
# OPEN DISTRIBUTED
# TELECOMMUNICATIONS ENVIRONMENTS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 16 mei 1997 te 16.45 uur.

door

Peter Leydekkers

geboren op 10 augustus 1965
te Hengelo (O)

Dit proefschrift is goedgekeurd door de promotoren:

prof. ir. E.F. Michiels
prof. dr. ir. L.J.M. Nieuwenhuis

Assistent-promotor: dr. ir. I.A. Widya

# Abstract

In the majority of European countries a twofold change is taking place in the telecommunications marketplace. Firstly, the traditional monopolistic state owned telecommunications provider is being privatised and secondly, the market in which this newly privatised provider operates is being opened to competition from other providers (some of them newly privatised themselves). The net result of these changes is an increase in the variety of services and the reduction in the cost of traditional services (commoditisation). To compete well, these new companies have to find a competitive edge. For new services this means that they have to be of a high quality, have advanced functionality and be produced in a short time frame. For existing services, the obvious choice is to compete on cost. The prices for services provided by ex-state owned companies, however, may be regulated in order to reduce their high existing market share, in which case the focus is more on customer care and quality. In addition, the convergence of the film and television market with video games, computer and telecommunications markets makes the picture very complex.

**Problem domain**

In this fast changing market a problem arises for Public Network Operators (PNOs) since their IT infrastructure is currently not yet ready to meet these requirements effectively. Their IT infrastructure is highly dedicated and existing systems are often monolithic and adapted to fit the supported business process. This prohibits the flexible exchange of information between different systems. Also, interoperability of services provided by computer, telecommunications and entertainment industries is difficult to achieve because there is a lack of cross-industry, standardised interfaces.

Standardisation organisations, such as ITU and ISO/IEC, and industrial consortia, such as Object Management Group (OMG), Digital Audio Visual Council (DAVIC), Internet Engineering Task Force (IETF) and Telecommunications Information Networking Architecture Consortium (TINA-C), have recognised the need for standards to enable interoperability in an open distributed environment. Substantial efforts have been put into the development of standardised architectures for the specification, implementation and deployment of distributed multimedia services. These efforts have only been partially successful for a number of reasons. Firstly, the development of standards has not kept up with the fast pace of changes in the increasingly competitive market. Secondly, competing and incompatible architectures have been developed. Thirdly, most architectures only address a subset of the problems at hand and several

architectures do not include multimedia at all in their architecture. Finally, there exists no comprehensive framework that addresses all relevant concerns and into which the various standards could be integrated.

## Objectives

Based on the problems identified above the main objective of this thesis is to describe an IT architecture that allows multimedia services to be easily specified and implemented in an open competitive telecommunications environment using both de-jure and de-facto standards. To achieve this objective, the thesis investigates the use of the Open Distributed Processing (ODP) reference model as an overall framework into which standards for distributed multimedia services in open heterogeneous processing environments can be placed. This involves the study of the compatibility of the ODP reference model with various standards. Additionally, a comparison of the approaches made by OMG, Microsoft, Bellcore, SPIRIT, OSF-DCE, IMA, TINA-C, DAVIC and ISO/ITU-T with respect to the ODP model are presented. At this stage in the investigation consideration is given to what refinements could be made to the ODP reference model to improve its ability to deal with multimedia services. These refinements are then validated by means of case studies.

## Results

An overview of the ODP reference model is given, explaining the basic concepts such as object orientation, distribution transparencies and the five ODP viewpoint languages. Having been involved actively in the creation of the ODP reference model for six years and working with the standard in KPN Research, it is concluded that ODP provides an ideal framework to position the different aspects of distributed systems. ODP-RM also provides insight into distributed processing and the concepts, rules and languages defined in ODP-RM are very useful for modelling and designing distributed systems. But in reality the ODP-RM standard is not being closely followed by industry which uses more de-facto standards (i.e. standards produced outside official standardisation bodies) due to the availability of concrete implementations.

In order to obtain insight in the de-facto standards for distributed processing an impression of the consortia involved is presented and their scope and mutual influence are highlighted. A comparison of the consortia is made based on their approach towards object orientation, separation principles used and multimedia concepts incorporated in their Distributed Processing Environment (DPE) specifications. This overview identifies the need for a general framework and discusses the role that the ODP standardisation process can play to combine the architectures of both de-jure standards (i.e. produced by official standardisation bodies) and de-facto standards. This kind of hybrid standardisation should lead to the provision of a long lasting architecture using components provided by the various consortia.

Focusing on multimedia in DPEs, the DAVIC, MiTV, IMA and TINA architectures are discussed for their support of multimedia services. The multimedia concepts found in

these architectures are positioned with respect to the ODP information, computational and engineering viewpoints to enable better comparison and future integration. From this investigation it is concluded that the various consortia define their own (specific) solutions and concepts. As expected, especially in the area of specific technology the solutions differ considerably. At a higher abstraction level (computational and engineering), similarity exists and only at this level common terminology is useful. This means that at the computational and engineering viewpoint, multimedia concepts should be standardised (providing interface descriptions, etc.). Each consortium should then do a particular mapping onto standards and specific technology.

Substantial work has been done during the preparation of this thesis to apply the ODP-RM computational and engineering views on the specification of and support for distributed multimedia services. In the computational language, the ODP concepts of 'stream interface', 'environment contract', 'binding object' and 'binding action' which were missing in early versions of the ODP standard are elaborated in this thesis. These extensions have been put forward to the ISO/ITU ODP standardisation group to be incorporated in the ODP standards as well as TINA-C deliverables during a one year involvement in the TINA-C core team in the USA. The correspondence between these computational concepts and engineering concepts such as 'stream interface reference', 'engineering channel' and 'engineering channel establishment' are also discussed. To validate these concepts a prototype of a multiparty multimedia binding object, useful for distributed services, has been built. To incorporate the specifications into multimedia platforms, additional standardisation is needed. This will allow the application designer to use complex multimedia binding objects as simply as operational bindings are currently used.

Besides functional aspects, this thesis also addresses quality aspects. To this extent, a general structure of the ODP environment contract concept is proposed using the TINA-C Object Definition Language (an extension of OMG IDL). It allows a specification of the QoS properties specific for multimedia applications in the computational viewpoint. Additional research is required to complete those QoS specifications in the five viewpoints and to establish the relationship between the viewpoints. In particular the relationship of the computational QoS specifications with the engineering specification (e.g. relation with system properties) is a challenging research topic.

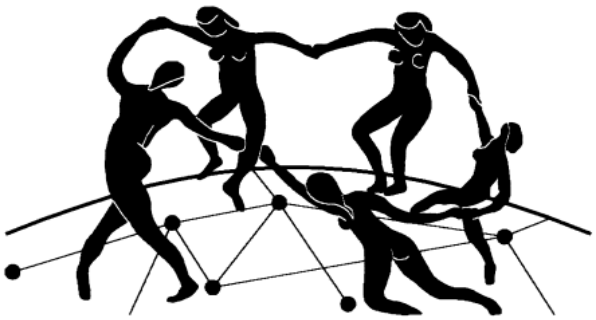
Finally, two case studies (multimedia conferencing service and synchronisation service) are presented to demonstrate the proposed multimedia extensions and methods introduced in this thesis. The case studies are based on prototyping work carried out in KPN Research. The TINA life cycle model is used as a framework (i.e. road map) to structure the case study specifications. Since the TINA life cycle model lacks detail it is completed with the ODP viewpoint languages, ITU-T study group 15, enterprise template and Object Modelling Technique (OMT) concepts.

# Acknowledgements

Writing a Ph.D. thesis can be compared to restoring an old-car. You start with nothing but pieces of metal and a lot of parts all spread over the floor and nothing to link them together logically. You have the image of the completed car in your mind and you are sure that one day you will have the image in reality but you don't know where to start. Restoring an old-car takes time if you want to do a proper job and this means not rushing parts that do not fit perfectly. A Ph.D. is similar in many ways, you write papers (parts of a car) and your first thought is that you can combine them and you will have a Ph.D. document. Wrong! What you end up with is a kit car that has no real charm. You have to force yourself to redo the job from scratch and make no reference to the papers, only in that way will you end up with a car where the parts fit smoothly. It is difficult to achieve such perfection alone and building an old-car with more people is also more fun, so is a Ph.D. Therefore I would like to thank several people. Firstly, my parents to whom I owe a debt of gratitude; in particular, for their encouragement and also for the skills of perseverance; the will to carry on no matter what which must come from my father, and advancement; the willingness to push myself to improve, which certainly comes from my mother. When building an old-car the first time, you need a technician that follows you all the way along until you can drive the car. Valérie Gay was this technician from the early stages until the test-drive of the car. She was there to guide me, show me how to fit the parts and provide the overall plan of how to build the car. Regularly, the process of rebuilding the car has to be checked to make sure it will drive in the end. Eddie Michiels, Bart Nieuwenhuis and Ing Widya were the car checkers and provided me with many useful indications to enable the car to be driven in the end. I also acknowledge KPN Research for providing me time and space for doing this job, especially John Ouderling and Bert Jan Teunissen who first introduced me to the possibility of doing a Ph.D. I would also like to thank Aart van Halteren and Martin Jacobs who contributed to this PhD by building prototypes as well as Leonard Franken, Frits Klok, Mike Schenk, Hans Hegeman, Niamh Quinn and Mark Bagley for checking parts of this thesis. A special experience was a year's stay in the USA and I acknowledge the TINA core team of 1995 for the interesting discussions on the future direction of the telecommunications industry.

Working full-time on a car is sometimes hard and luckily I had nice people around me to support me. I'd like to thank Ellineke Bolt, Irene Kwaaitaal and Michel van den Bempt in particular, although there are many others.

# Contents



# 1 Introduction

*This chapter gives a brief overview of the evolution of information technology (IT). IT has evolved from monolithic proprietary systems to a worldwide network of interconnected systems, which exchange all kinds of information, ranging from simple text to sophisticated audio-visual information. The telecommunications industry, for many years untouched by the changes going on in the area of information technology, now has to face reality. Indeed, in rapid pace the telecommunications industry is now adapting its infrastructure to prepare for an open market. The telecommunications infrastructure is migrating towards open distributed processing environments that offer a broad spectrum of services which go far beyond the traditional telephony oriented services. This thesis is positioned in the centre of the merging of IT and telecommunications industries into a single market. This evolving world is approached from a standardisation point of view since the author believes that standardisation is necessary to offer a generic solution that is acceptable to both the telecommunications and computing industry. This chapter discusses and motivates the scope of this thesis, introduces the terminology used in this thesis and presents the approach followed.*

## 1.1 A changing world

Information technology is one of the fastest changing areas in our society. In less than 40 years, the world has seen an evolution from huge and expensive computer systems, with limited capabilities, to small and powerful compact personal computers. Nowadays, these systems can be interconnected worldwide providing multimedia services at low cost. In parallel, the telecommunications industry is facing a dramatic change from static, monopoly oriented companies, which were slow in introducing new

types of telephony services, to customer oriented commercial companies, which are introducing all kinds of sophisticated telecommunication services in a highly competitive market. Such a dramatic change would not have been imagined 40 years ago and the end of this evolution is still uncertain.

### From monolithic to distributed environments

The computer systems in the early 60's were monolithic mainframes and applications were written in low level programming languages specific to the operating system used. The hardware and software were proprietary and communication with other computer systems was out of the question. This resulted in islands of monolithic mainframe systems with highly specialised non-portable applications. Users were directly connected to the mainframes through simple terminals [1][2][4].

In the 70's, mini computers were introduced and the first proprietary computer networks that connected the terminals to the mainframes came into the market. Also the terminals became more intelligent and various functions were now locally available in the terminal, for example graphical functions to build up a screen. However, the applications were still situated in the mainframe computer (see Figure 1.1).

In the 80's, this situation changed when the first generation of client/server applications was introduced. The functionality provided by the monolithic systems was split into a client and server component and spread over a variety of computing systems. This change was possible due to the decreasing cost of CPU power and the availability of local area networks. The applications were still monolithic, only system services such as printing servers and database servers were distributed. This generation of client/server applications appeared in the early 80's when companies still had relatively simple development and run-time environments using only one operating system, one database management system and one network protocol. This resulted in semi-isolated LANs. This generation reached its end in the mid '80s partially because of the introduction of the personal computer which causes companies to deal with a broader variety of operating systems, networking software and database management systems [3][4][5].

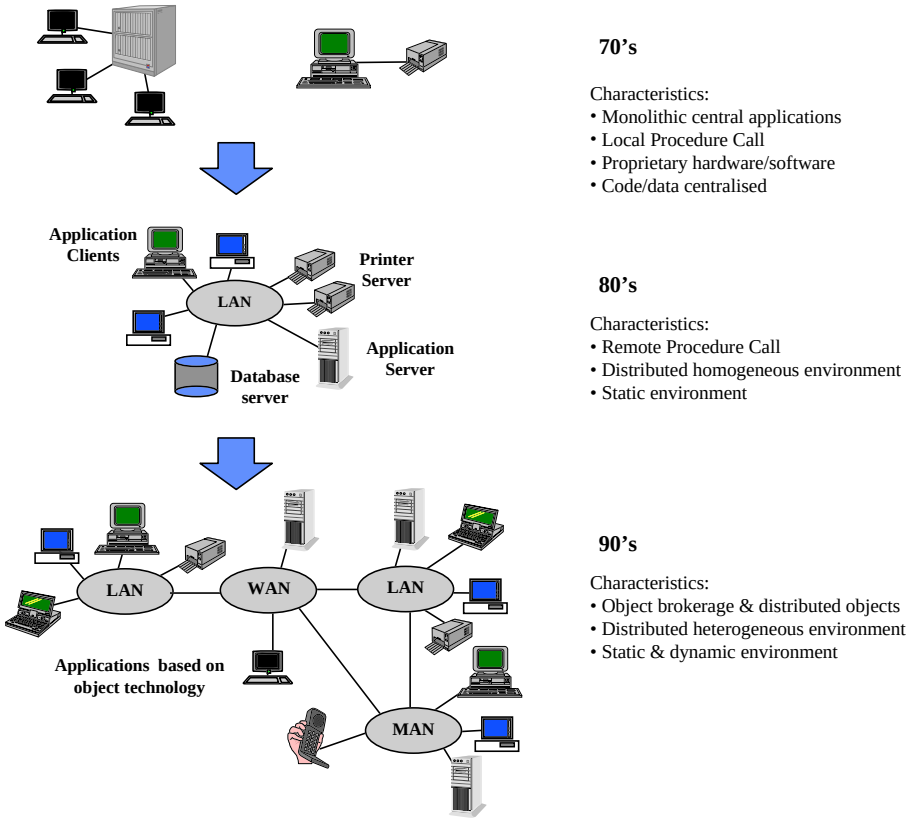



Figure 1.1: Evolution from monolithic to distributed object oriented environments

Nowadays, we are facing the second generation of client/server applications [3][4][5]. This generation appeared because of the need to interconnect islands of computer systems. A shift occurred from single server, Local Area Network (LAN) based client/server systems to an infrastructure of client/server applications where each machine on the networked infrastructure can be both a client and a server. This shift is made possible due to the increased availability of low-cost bandwidth on Wide Area Networks (WAN) and a new generation of network-enabled multithreaded desktop operating systems [2]. Also the popularity of the Internet and on-line services contributed to the fast expansion of this new client/server generation to the home environment enabling millions of people to have access to a huge amount of services.

## Distributed object technology

Many client/server applications are built upon middleware that encompasses the mechanisms to exchange data in a distributed environment. Although a good step forward, these client/server applications are still difficult to build, manage and extend [1][2]. This has been recognised by the computer industry (e.g. OMG, Microsoft) and telecommunications industry (e.g. TINA [6], EURESCOM [28]). It is now commonly

agreed that distributed object technology is a very promising approach to solve the problems mentioned above. With distributed object technology the advantages of object orientation and distributed computing are combined. It provides advantages such as portability (applications deployed on different platforms), interoperability (communication across different environments), coexistence (with legacy applications through object wrappers) and extensibility (add functionality on demand). With distributed object technology the data and logic are encapsulated within objects, which easily facilitates objects to be located anywhere in the distributed environment. Applications developed using distributed object technology do not prescribe to a specific environment on which applications have to be deployed. It can be a fully distributed system, a centralised system or a combination of both.

Although distributed object technology is promising, in practice however, it will be unlikely that all organisations will use distributed object technology as the single solution for the modelling and implementation of distributed applications. Also competing de-facto standards have been developed for distributed object technology which prohibits simple interoperability of services that are based on different standards. If we want to achieve a global distributed environment of networked computers, without running into chaos; there is a need for an overall reference model which can integrate the various de-facto standards to ensure interoperability of systems and portability of applications.

### Telecommunications and distributed environments

Distributed environments are inherent for the telecommunications industry. The interconnected telecommunications networks can be regarded as the largest online open distributed system that has ever been built. Distributed systems are the combination of computing systems and the (tele)communication networks that interconnect them. For decades, Public Network Operators (PNOs) offered the relatively straightforward service of standard voice connection. However, this situation is changing for PNOs due to de-regulation, globalisation and liberalisation, which increase the competition in the telecommunications market. It forces PNOs to offer existing (and new) services in a fast, cost effective way. PNOs are forced to adopt a market oriented approach and must evolve from a limited service offering in a single country to a portfolio of various services which can be tailored to customer needs.

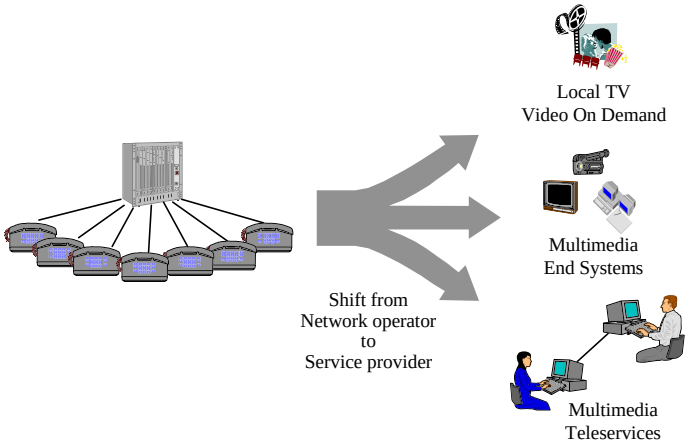



Figure 1.2: Shift from network operator to service provider

To face the competition and to consolidate, PNOs want to expand their service portfolio [10] to non-traditional telecommunication markets. For example, Internet related services (e.g. e-mail and worldwide web) offer new opportunities since Internet has brought increasing diversity and sophistication of telecommunication services available to the computer desktop. Services with a strong multimedia component will certainly penetrate the business environment but the most interesting market-segment for multimedia services is the home environment [9].

Another tendency is that the customer wants to have certain influence on the network resources he/she uses for a particular service. The customer may want to request certain network resources such as bandwidth according to his/her demands. Issues such as dynamic bandwidth reservation and re-negotiation, as well as, on-line billing information will play an increasing role for the new services offered by PNOs.

However, a problem arises since the existing IT infrastructure of PNOs is not very well suited to meet the requirements outlined above. The IT infrastructure can be characterised as a collection of proprietary, often incompatible systems which use non-standardised solutions to exchange information. The lack of a simplified and evident IT infrastructure for PNOs resulted in many legacy systems where overlap exists in their functionality and information. To develop, introduce, maintain and extend telecommunication services is expensive and time consuming. To solve these problems, organisations such as the TINA-consortium [6] exist to define solutions which take into account telecommunications specific developments (e.g. TMN, IN), and developments from the computing industry such as client/server architectures. An open issue is, however how to deal adequately with multimedia services which have strict QoS and performance requirements.

### Multimedia and distributed environments

An increasing number of companies (e.g. public utilities companies, TV cable companies, and film industry) will provide multimedia[1] services where telecommunication is important amongst the services provided [7]. An example would be cable network operators who enhance their cable network with two-way communication to provide multimedia services such as teleshopping. The availability of multimedia personal computers and set-top boxes for television provide multimedia applications to the home environment at a reasonable price. It is expected that the first mass multimedia services offered to the home environment would be Worldwide Web services (WWW) and multimedia mail services. The services offered to the end-user will allow customisation and personalisation. This means that the end-user can select those types of services he/she wishes to use and may configure the selected service to his/her own preferences.

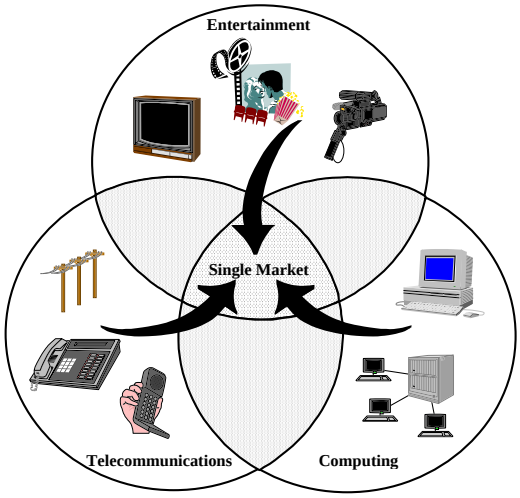



Figure 1.3: Integration of telecommunication, computing and entertainment into a single market

The convergence between telecommunications, computing and entertainment causes new problems due to the heterogeneity of the systems used and services offered by these companies. Interoperability between services is difficult to achieve due to the lack of open interfaces, which prohibits a customer to use the services offered by different companies in a transparent manner. An open distributed environment, which meets the requirements of all stakeholders, is necessary and standardisation of (multimedia) interfaces needed.

1 'Multimedia information' in the context of this thesis does not only refer to a single information type such as text but a combination of information types such as audio, video and still graphics (see Section 1.4 for a precise definition)

### Standardisation and distributed environments

Numerous attempts have been made to provide generic (or standardised) solutions for parts of distributed systems. International standardisation bodies such as ISO/IEC and commercial efforts grouped in industrial consortia tried to standardise their solutions. Each standardisation effort focused on credible market opportunity for each player involved but often these standardisation activities came too late, were not powerful enough or different (competing) solutions for the same problem were present [2]. The market also forced the companies to come up with products prior to the (full) availability of standards. Together, this has lead to divergent solutions for the Distributed Processing Environment (DPE) exemplified in different solutions for computing platforms, interfaces, tools and networks. It is now commonly admitted that there will be always a heterogeneous and changing environment where multiple (de-facto and de-jure) standards coexist.

However, continued growth of distributed services in a worldwide perspective depends on the convergence of these de-facto and de-jure standards into common goals and plans for the evolution of the DPE [12]. To meet these goals, ISO/ITU-T has an ongoing standardisation activity, which tries to integrate several aspects of distributed systems in a single international standard: the Open Distributed Processing - Reference Model (ODP-RM). An issue is whether ODP-RM is really suitable to combine both de-facto and de-jure standards into a consistent set of standards for open distributed processing.

## 1.2 Scope and objective of this thesis

The scope and objective of this thesis are derived from the trends and observations described in the previous section and are reflected in the title of this thesis: ***"Multimedia Services in Open Distributed Telecommunications Environments"***.

The term *'Distributed environment'* in the thesis title is to be interpreted here as the infrastructure that provides the processing environment for distributed services. This infrastructure is the combination of computing nodes and a communication network that interconnects the computing nodes. *'Open'* in this thesis should be interpreted to signify those distributed environments that permit many (often competitive) stakeholders to coexist. Open also implies a heterogeneous environment comprising of a mixture of LAN, WAN and MAN networks and/or heterogeneous equipment. The qualification *'Telecommunications'* is used in this thesis to signify that the public telecommunication network is involved and issues such as billing and connection management across different authority domains are important. Also the multimedia services described in this thesis have their roots in a telecommunications environment.

The area of '*open distributed telecommunications environment'* is very broad and covers many issues ranging from interconnecting different types of networks to the commercial introducing of new services in a distributed environment. Only certain issues can be addressed in the time frame of a Ph.D. The author expects that *multimedia*

*services* will be an important category of services for the future distributed environment. Insight into the modelling of services in open distributed environments, which have a real-time component, is important for the specification and development of the new generation of information and telecommunications services. The objective of this thesis can be formulated as follows: Describe an IT architecture that allows multimedia services to be easily specified and implemented in an open competitive telecommunications environment using both de-jure and de-facto standards. This overall objective is refined in the following sub-objectives:

- Provide insight into the ODP reference model and investigate whether ODP-RM can be used as an overall framework to support distributed multimedia telecommunications services for open distributed processing environments;
- To study how the ODP reference model and de-facto standards can be used together for modelling and implementation of multimedia telecommunication services;
- Propose refinements of ODP concepts related to multimedia applications that are also useful for the telecommunications environments as defined by the TINA consortium;
- To validate the multimedia refinements proposed in this thesis. Therefore, two case studies are described that show the usefulness of a combination of ODP-RM; de-facto standards and proposed multimedia extensions to design distributed multimedia services.

**Limitations**

The architectural parts of DPEs are focused on and abstracted from detailed implementation issues. This means that technical issues such as Application Programming Interfaces (APIs) or protocol implementations suitable for DPEs shall not be focused on. Changes in the area of DPEs occur so fast that a particular technical solution published in this thesis will be soon out-of-date. In contrast, modelling concepts have a longer lifetime if related to architectures that are well accepted in both the telecommunications and computing domains. Also, the author does not want to create yet another design method for telecommunications services but use existing ones instead and refine them where necessary.

## 1.3 Approach

The concepts, rules and languages defined in ODP-RM are used in this thesis as the basis for modelling and designing distributed systems. Since ODP-RM is a general framework, specific architectural frameworks such as TINA and OMG's OMA are used to refine the description of multimedia concepts of a DPE and the modelling of distributed telecommunications services. An overview of the various industrial bodies, which are involved in the standardisation of distributed processing and give insight in various solutions available for the DPE is provided. To increase acceptance of the

proposals developed in this thesis, it should stay close to the architectures developed by these standardisation bodies.

The focus is then on the ODP computational and engineering viewpoint, and ODP concepts for multimedia are detailed. The aim here is to demonstrate how the various multimedia concepts described in de-facto standards can be integrated into the ODP-RM framework.

Two case studies are presented to show the practical usage of the proposed multimedia extensions to ODP. The *Multimedia conferencing service* and *Synchronisation* case studies illustrate the concepts and methods introduced in the thesis. For the modelling approach, OMT together with more telecommunications specific methods such as the TINA-C service life cycle model is applied. In addition, the ODP viewpoints and concepts are used to complete the design of the case studies. The case studies are based on prototyping work carried out in KPN Research.

## 1.4 Terminology related to distributed environments

In the previous sections, the term 'distributed environment' is used several times to denote a mixture of computing and telecommunication systems that are interconnected for the exchange of data. Development of an unambiguous and precise definition of 'distributed environment' is a difficult task, and depends on the user's context that uses this term. In [13] an overview and discussion are presented of the various interpretations of experts related to distributed environments. Based on this discussion, Figure 1.4 represents a typical distributed environment that fits many interpretations and is commonly used in consortia such as TINA-C and OMG. Figure 1.4 also illustrates and relates the different terminology used in the context of this thesis.

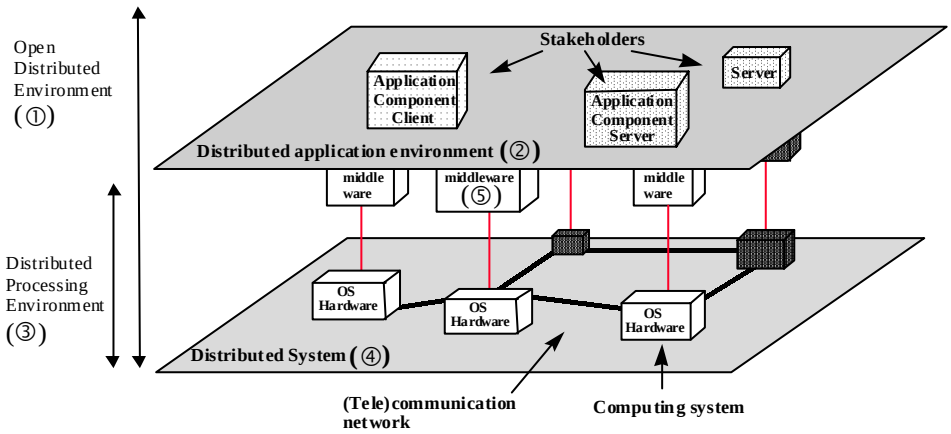



Figure 1.4: Terminology related to Open Distributed Environment

The *Open Distributed Environment* supports a particular *architecture*. It can be characterised by its *openness* and the *distribution transparencies* it offers. The applications in the distributed environment may encompass *multimedia* concepts. An open distributed environment can be decomposed into two environments that are the *Distributed Application Environment* and *Distributed Processing Environment*. These concepts are clarified below.

### Architecture

An open distributed environment (Figure 1.4, ①) is a highly complex system, and an architecture is required to structure such a system into manageable (interrelated) components that can be handled independently. An architecture is defined in this thesis as composed of:

- *Concepts* (or terminology) for the modelling of the components that make up an open distributed environment. The components can be recognised as elements in the real world system;
- A set of *structuring rules* describing how to use these concepts to specify the components. It also defines the interrelationship between the components of the open distributed environment.

A similar definition of architecture is found in ODP-RM and TINA. An architecture according to [28] also includes the modelling approach followed for the analysis and design of a distributed environment. Also a modelling approach is included as part of an architecture in this thesis and the author focuses on those architectures that are based on *object orientation*. The basic principles of abstraction, encapsulation and composition in object orientation make it suitable to deal with complex and large systems that are distributed. Object orientation is also suitable for the design of distributed applications [28].

### Openness

Openness is a general term. In this thesis the following definition of openness related to open distributed environments is used:

- Openness in the context of *Distributed Application Environment* (②) means that multiple (competitive) stakeholders can be involved and that the distributed environment is not limited to a selected (closed) group;
- Openness in the context of *Distributed Processing Environment* (③) contributes to the portability and interworking of application parts in a DPE. *Portability* is the ability to execute an application component in a heterogeneous environment without modification. Heterogeneity includes equipment heterogeneity, operating system heterogeneity and authority heterogeneity (e.g. co-operation between autonomous network operators). *Interworking* is the ability to support meaningful interactions between the application parts possibly residing in different computing systems.

### Distributed Processing Environment (DPE)

The terms 'Distributed Computing Environment' and 'Distributed Platform' are considered as synonyms for Distributed Processing Environment (③) in this thesis. The DPE encompasses *distributed system* (④) and *middleware* (⑤).

- The *distributed system* is the combination of *computing systems* and the *(tele)communication networks* that interconnect them. The *computing system* consists of the operating system and the hardware of a machine. The *(tele)communication network* refers to a (possibly) integrated configuration of LAN, MAN and WAN types of networks, which are able to convey information between computing systems that are geographically distributed;
- The *middleware* is the software that supports distributed processing where (part of) the applications may be distributed over several computing systems. Middleware makes the application code independent from the distributed system. Excluded in the definition of middleware is any product or tool whose prime purpose is other than to provide connectivity. Middleware enables interoperability in the sense that its purpose is to connect application processes, which are normally expected to be remote from each other. Middleware has evolved from the need to spread processes across heterogeneous networks of computing nodes. Middleware extends the traditional operating system interprocess communication mechanisms [2][3][4][5].

### Distribution Transparencies

Distribution transparencies are properties offered by a (distributed) system to hide effects caused by distributed applications. Transparencies are mechanisms that enable the application designer to work in a world, which is transparent to a particular concern. Application designers simply select which transparencies they want and where in the design they are to be applied. The application components (or objects) using the distributed system may use certain transparencies they need and handle other aspects of distribution characteristics as they want. Location transparency is a well-known transparency, which hides the need for an application component to have information about the location of an other application component in order to invoke an operation. In Section 2.2.2 the various transparencies as defined by ODP-RM are detailed.

### Distributed application environment

The *distributed application environment* (②) is the environment where services are provided to the distributed applications.

- A *distributed application* consists of several application components, which can be physically located on several computing systems in a DPE. Client/server applications are an important category of distributed applications. With client/server applications certain application components (or objects) behave as clients, request information (e.g. database query) and/or processing, and other application components act as servers, responding to those requests;

- A *service* is a meaningful set of capabilities offered to its environment [50]. A service in a distributed application environment can be offered by a set of possibly distributed applications. In an object oriented environment an object offers services (an object, which makes a function available, is said to offer a service). The functions and services are specified in terms of the behaviour of the object and of its interfaces.

## Multimedia

The terms multimedia, multimedia services, distributed multimedia systems etc., often appear in the literature, having a specific meaning depending on the context of the user of these terms. This section defines several terms related to multimedia, as they will be used in this thesis.

- *Media:* can be a simple data encoding format (e.g. ASCII text) or it can be more complex such as a movie that defines an audio encoding format, a video encoding format and their relationship in a stream [33];
- *Static media:* a media for which time is not intrinsic, such as text [33]. Synonym is *discrete data*;
- *Time-based media:* a media for which time is intrinsic, such as audio, video and animation graphics. A media is time-based whether it is stored or live. [70]. Synonym is *continuous media*;
- *Multimedia information:* information composed of time-based media and possibly static media [33]. It should be noticed that by default, when one speaks of multimedia it is understood that at least one of the media is time-based. A document composed of text and photos is therefore not classified as ‘multimedia information’ in the context of this thesis;
- *(Distributed) multimedia system:* A (distributed) system has the ability to simultaneously capture, process, generate, present and transport multimedia information with possible spatial and temporal relationships between multiple (time-based and static) media;
- *Multimedia service:* A meaningful set of capabilities handling multimedia information that is offered to its users;
- *Multimedia application:* An application that handles multimedia information;
- *Stream:* time-based media, which flows from a source virtual device to a sink virtual device [33];
- *Real-time application*: is an application that requires multimedia information delivery for immediate consumption. A telephone conversation is an example of a real-time application. In contrast, a non-real-time application requires that the information is stored (maybe temporarily) at the receiving endpoint for later consumption. An electronic mail is an example of a non-real-time application.

The quality of time-based flows is an important characteristic of multimedia services. Quality of service (QoS) is used in this thesis according to Vogel [57], and is defined as:

- *QoS*: represents the set of those quantitative and qualitative characteristics of a distributed multimedia system necessary to achieve the required functionality of a multimedia application [57]. *Quantitative* characteristics can be measured such as bits/s, task processing time etc. *Qualitative* characteristics are closely related to the user perception of a distributed system. It describes the expected services offered by the system such as inter-stream synchronisation, ordered delivery of data and scheduling mechanism [58].

Vogel's definition of QoS is more precise than the ODP-RM definition of QoS, which defines QoS as 'a set of qualities related to the collective behaviour of one or more objects' [15].

## 1.5 Structure of this thesis

This thesis consists of 6 main chapters that are related to each other as shown in Figure 1.5.

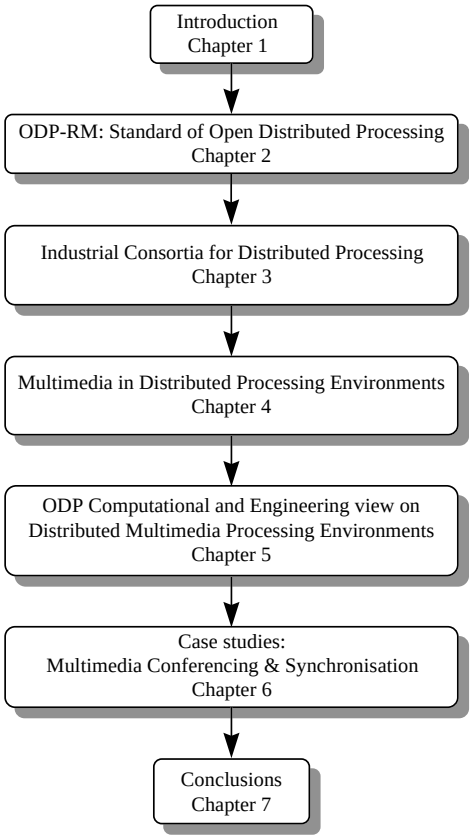



Figure 1.5: Relationship between the chapters

Chapter 2 provides an overview of the ODP reference model, explaining the basic concepts such as object orientation, distribution transparencies and the five ODP viewpoint languages. This chapter explains the ability of ODP-RM to tackle the design of complex distributed systems. ODP-RM is used as the basis for this thesis and applied in the other chapters.

Chapter 3 describes the results of several relevant industrial consortia. This chapter provides background information that attempts to present to the reader an impression of the ‘forces’ at play. It also shows that ODP-RM could serve as a reference for a family of architectures on which distributed applications will be based.

Several consortia mentioned in Chapter 3 specify a distributed processing environment to support distributed services. In Chapter 4 the author focuses on four of them, i.e. TINA, IMA DAVIC and Microsoft. They have the potential of being accepted by a large community in both the telecommunications and the computing industry. The purpose of Chapter 4 is to describe and analyse these architectures focusing on multimedia components. The multimedia concepts found in these architectures are positioned in the ODP viewpoints in order to compare them.

The aim of Chapter 5 is to describe the use of ODP-RM computational and engineering views for the specification and support of multimedia real-time communication. In Chapter 5, the author proposes additions and refinements of multimedia modelling concepts. In particular, this chapter addresses Quality of Service (QoS) specifications for time-based dataflows and refines ODP concepts such as stream interface and binding.

Chapter 6 presents two case studies, based on concepts presented in the previous chapters. The first case study: 'Multimedia conferencing service' illustrates the design of a particular multimedia service. The second case study: 'synchronisation of streams' illustrates the design of a component that deals with real-time multimedia synchronisation.

Chapter 7 presents the conclusions and discusses the author's view on the use of de-facto and de-jure standards for the specification of distributed multimedia services. Also open issues for further study are addressed.

# 2 ODP-RM: Standard of Open Distributed Processing

*This chapter provides a detailed explanation of ODP-RM. It describes the motivation for a standard for open distributed processing and the foundations of ODP-RM which are object orientation, distribution transparencies and ODP viewpoints. The major concepts and rules for each of the five viewpoint languages of ODP-RM are described. The objective is to provide the necessary information on ODP-RM that is used as a basis for the remaining part of this thesis. It also presents the author's view on the ODP-RM architecture and standardisation process.*

## 2.1 Introduction

In the eighties, ISO/IEC and the ITU-T investigated the problem of interconnecting heterogeneous systems resulting in the Open Systems Interconnection Reference Model (OSI-RM). This reference model focuses primarily on the standardisation of protocols needed for interconnecting systems. However, the distributed systems community identified the need to address the standardisation of *inter-communication* between applications rather than only *inter-connection* applications. Applications operating in a heterogeneous distributed environment have additional requirements such as openness, integration and inter-operability. A more application-oriented model is needed to address all aspects of the distribution of applications.

ISO became aware of this need and initiated a new standardisation activity called Open Distributed Processing (ODP) in 1987. Shortly afterwards, the ITU-T initiated a new

study question called Distributed Application Framework (DAF), the purpose of which is to create a framework for distributed applications. The goals of both initiatives are very similar and from the start ISO and ITU-T collaborated and decided to publish a joint standard: the Open Distributed Processing - Reference Model (ODP-RM) [14][15][16][17].

### Objectives of the ODP-RM standard

Distributed systems will come to the market in many different forms and it is unlikely that a single standard will cover all applications of distributed systems (see the various consortia providing different solutions described in Chapter 3). The rapid growth of distributed processing systems introduces new problems since applications and services in such systems are distributed by nature and the hardware and software support is heterogeneous (a variety of equipment, operating systems, languages, database models, management authorities etc.).

A framework is required which covers all relevant aspects of distributed systems to co-ordinate the development of the various standards needed for open distributed systems. Such a framework describes and relates the functions that make up a distributed system without going into implementation details. The ODP reference model is expected to provide a general framework that addresses distributed systems, which have to operate in a heterogeneous environment.

The objective of ODP is to enable seamless interworking of distributed application components regardless of various forms of heterogeneity. To achieve this objective, the first step in the development of standards for ODP was the creation of the ODP reference model. The ODP-RM provides concepts, rules and models for building distributed systems. The ODP reference model defines several important foundations that are used throughout the standard. The most apparent foundation is the use of *object orientation* for the specification of distributed applications and their components. In addition, ODP-RM uses two abstraction mechanisms to deal with the complexity of distributed systems; these are *'distribution transparencies'* and '*ODP viewpoints'*.

The objective of this chapter is to provide insight into the most important modelling concepts available in ODP-RM.

## 2.2 ODP foundations

This section explains the foundations of ODP-RM, that is, object orientation, distribution transparencies and the five ODP viewpoint languages.

### 2.2.1 Object Orientation

ODP specifications are based on the concept of objects to model problem domain entities. Objects can represent various things depending on the problem domain that is modelled, and the abstraction level (or viewpoint) that is applied. Objects can represent a 'real-world entity', an 'idealised entity', the 'subject of concern' and so on. Objects in ODP can be of arbitrary granularity. Objects are entities that contain information and offer services. A system is modelled by a set of interacting objects. An object is characterised by its identity, encapsulation, abstraction and behaviour, which make it distinct from others. Each of these concepts and derived concepts are detailed below.

#### Encapsulation and abstraction

*Encapsulation* is the property that the information contained in an object is accessible only through interactions at interfaces supported by the object. Encapsulation provides the effect of *abstraction* implying that internal details of an object are not visible to other objects. A change in the state of an object is only possible as the result of an internal action or as the result of an interaction with its environment (encapsulation ensures that object interaction has no hidden side effects). Interaction with one object cannot affect the state of other objects without additional interaction(s). Objects can only interact at interfaces, where an interface represents a part of the object's behaviour related to a particular subset of its possible interactions. An object defines a set of services that can be offered to the clients of the object through the interface(s) supporting it.

#### Behaviour/state

The *behaviour* of an object is defined as the set of all potential actions in which an object may take part. The ODP object model does not constrain the form or nature of object behaviour. State and behaviour are interrelated concepts. *State* characterises the situation of an object at a given instant. The behaviour of an object describes all of the object's potential state changes. The current state of an object is determined by its past behaviour. Conversely, potential actions that an object may undertake are determined by its present state and will also depend on the actions in which the environment is prepared to engage. The behaviour exhibited at the interface of an object can be a subset of the total behaviour of the object.

#### Interfaces

An *interface* is the only means to access an object. An interface consists of a set of interactions, which can be signal, stream or operation interactions (see Section 2.3.3.1 for details). An ODP object can have multiple interfaces, which allow an object to organise the total set of interactions supported by the object into subsets. The support for multiple interfaces allows, for example, to separate management and non-management operations into different interfaces, at which different QoS characteristics (e.g. different security levels) can be associated. To use an interface, it is necessary to

uniquely identify it within the context of the ODP system. Interface identifiers are used for this purpose. They can be passed on between objects that want to use the interface. Once an interface has been identified, particular interactions can be identified within the context of that interface.

### Type and class

A *type* is a property of a collection of objects. An object satisfies 'a type', or is 'of the type', if the property holds for that object. Objects do not have to be similar to satisfy the same type; they only need to possess the properties prescribed by the type. Types implicitly classify objects into sets known as classes. A *class* is the collection of all objects associated with a given type. The notion of type is very general and can be specialised in various ways. It is useful in several contexts where it is necessary to verify properties of objects.

### Polymorphism

*Polymorphism* is the property that the same operation can do different things depending on the class that implements it. Objects belonging to different classes can receive the same request but react in different ways. The initiator is not aware of this difference and the receiver object interprets the operation and provides the appropriate behaviour. Polymorphism allows one to view similar objects through a common interface and eliminates the need to differentiate between the objects. It should be noted that ODP does not explicitly refer to polymorphism but from the descriptions in [15] it is concluded that polymorphism is supported by the ODP object model.

### Inheritance

*Inheritance* is the mechanism to create child classes (also known as subclasses) from a parent class. Child classes inherit operations and data structures from the parent class. New operations can be added to the child class or can even override inherited operations to define new behaviours. The parent's operation is not affected by this modification. ODP supports the inheritance principles through their description of 'class and subclass hierarchy' [14].

### Templates

Templates are used to describe objects. A template describes common features of similar objects (i.e. objects of the same type). For instance, it can describe state parameters and operations. A template contains sufficient information to instantiate a new object from it. So, the template characterises the set of objects instantiated from it. Typically, this entails establishing the initial state of the initiated object. For example, a buffer object might be created (from a buffer template) with empty contents.

### 2.2.2 Distribution transparencies

Table 2.1 shows the distribution transparencies as defined in ODP-RM [16].

| Transparency | Masks | Effect |
|---|---|---|
| Access | Masks the difference in data representation and invocation mechanisms to enable interworking between objects. | Solves many of the problems of interworking between heterogeneous systems. |
| Failure | Masks the failure and possible recovery of other objects (or itself) to enable fault tolerance. | The designer can work in an idealised world in which the corresponding class of failures does not occur. |
| Location | Masks the distribution in space of interfaces. Location transparency for interfaces requires that interface identifiers do not reveal information about interface location. | Provides a logical view of naming, independent of the actual physical location. |
| Migration | Masks the ability of a system to change the location of that object. | Migration is often used to achieve load balancing and reduce latency. |
| Relocation | Masks the relocation of an interface from other interfaces bound to it. | Allows system operation to continue when migration or replacement of objects occurs. |
| Replication | Masks the use of a group of mutually behaviourally compatible objects to support an interface. | Enhances performance and availability of applications. |
| Persistence | Masks from the object the deactivation and reactivating of other objects (or itself). | Maintains the persistence of an object when the system is unable to provide it with processing, storage and communication functions continuously. |
| Transaction | Masks the co-ordination of activities amongst a configuration of objects to achieve consistency. | Provides consistency guarantees about interactions between applications. |

Table 2.1: Transparencies

It should be noted that ODP-RM is not extensive with respect to distribution transparencies. Additional distribution transparencies have been identified for distributed systems, which can also be useful. For example:

- Concurrency transparency where several users or application programs can concurrently use shared data without mutual influence;
- Scaling transparency that allows the system to grow without changing the overall system architecture or application programs.

### 2.2.3 ODP Viewpoints

To deal with the complexity of a distributed system, ODP uses a framework of abstractions that considers the system from a set of interrelated viewpoints. Each viewpoint represents a different abstraction of the original system. ODP identifies and uses five viewpoints. These are called the enterprise, information, computational, engineering and technology viewpoints. A viewpoint is a representation of the system with the emphasis on a specific concern while ignoring other characteristics that are irrelevant for that viewpoint (Figure 2.1). Certain viewpoints are useful for the

description, analysis and synthesis of the concerned systems whereas other viewpoints are primarily concerned with the objectives, realisation and behavioural descriptions. For example, a computational model focuses on the object interfaces and the interactions that occur at these interfaces, while an information model focuses on object classes, the semantic relations among them. Physical distribution is only visible in the engineering and technology viewpoints, which focus on the structure and implementation.

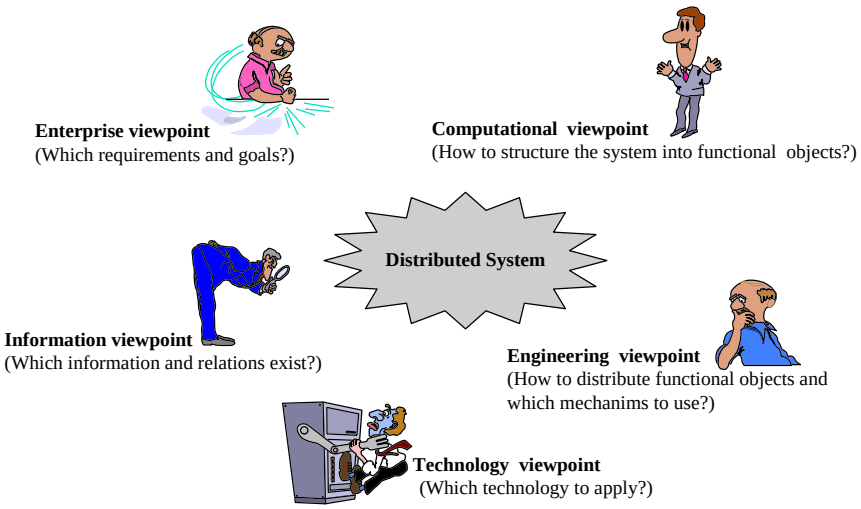



Figure 2.1: ODP-RM Viewpoints

The five ODP viewpoints can be characterised as follows:

- The *Enterprise viewpoint* focuses on the requirements, the purpose and policies that apply to the specified system independent of distribution aspects that might be applicable to the system. It covers the business aspects and the human user roles with respect to the system and the environment with which the system interacts. From the enterprise viewpoint the overall objectives of an ODP system are seen;
- The *Information viewpoint* is concerned with the information that needs to be stored, exchanged, and processed in the system of concern. The information viewpoint describes the information model of the system and of the individual components identified. It provides a common view, which can be referenced by the specifications of information sources and sinks and the information flows between them;
- The *Computational viewpoint* is concerned with the description of the system as a set of interacting objects, and describes how distributed applications and their components are structured in a distribution transparent way. This implies that the structuring of applications is independent of computers and networks on which they run. This viewpoint specifies the individual, logical components, which are the sources and sinks of information. The model used in the computational

viewpoint is object based and a distributed application consists of a collection of computational objects;

- The *Engineering viewpoint* focuses on the infrastructure required to support distributed processing. It is concerned with the distribution mechanisms and the provision of the various transparencies needed to support distribution;
- The *Technology viewpoint* focuses on suitable technologies to support the implementation aspects of the distributed system. It is concerned with the implementation details of the components from which the distributed system is constructed.

Different viewpoints address different concerns, however the viewpoints are not orthogonal, i.e., they have some dependencies. They are each a projection of the complete system specification.

## 2.3 ODP viewpoint languages

In order to represent an ODP system from a particular viewpoint it is necessary to define a structured set of concepts to construct a specification in the concerned viewpoint. Such a specification constitutes a model of a system in terms of these concepts. The set of concepts provides a language that is specific for each viewpoint. This language is the means to create a model of the problem domain from the concerned viewpoint. A viewpoint language is not defined to replace existing languages. Its purpose is to specify concepts in terms of which specifications of a system from that viewpoint must be structured. Thus in principle any existing language can be used for the specification of a system from a particular viewpoint, under the condition that those specifications can be interpreted in terms of defined viewpoint concepts. The viewpoint languages could be related to designer roles as shown in Table 2.2.

| Viewpoint | Designer role | Models and specifies | Example language/support |
|---|---|---|---|
| Enterprise | Analyst, Business analyst | User requirements, policies | E/R diagrams |
| Information | Information Manager or Database Administrator | Information of concern for the distributed system | OMT, Z |
| Computational | Application Designer | Description of functional objects and interfaces - Requirements on the engineering support | IDL, TINA-ODL |
| Engineering | Developer, Operating system, database or communication expert | Mechanisms to support the computational specification | ORB |
| Technology | System or Electronics engineer | Technological support, implementation issues | Chorus, Mach, ATM, TCP/IP |

Table 2.2: ODP viewpoints languages

A viewpoint language consists of a set of *concepts* and *rules*. Concepts are the vocabulary used for a specific viewpoint language for the description of a distributed system. Rules describe the grammar for the composition of the distributed system for a

particular viewpoint using the associated concepts. A *viewpoint specification* of a system uses the concepts of that viewpoint and satisfies the structuring rules of that viewpoint.

Sections 2.3.1 to 2.3.5 describe the enterprise, information, computational, engineering and technology viewpoint languages. A summary of important concepts and rules is given for each viewpoint language. The concepts are illustrated by examples where appropriate.

### 2.3.1 Enterprise language

The enterprise specification provides a description of the requirements and objectives that the environment imposes on the system to be designed. It justifies the existence of the system. The concepts and rules described in the enterprise language are rather general and do not constrain too much the analysis of the system. Its purpose is basically to describe the boundaries of the system. Different notations for the enterprise specification can be used to support specific organisational structures and business practices. ODP-RM is not prescriptive in this viewpoint, and only requires that an enterprise specification is created. The enterprise specification identifies the requirements, policies and actions that are required by the system and the positioning of the system into its environment, identifying the stakeholders that are involved. The enterprise specification can be composed of the combination of the following statements [18]:

- What is the purpose of the system (or its components) in the organisation?
- What is the scope of the system in the organisation?
- What policies are applicable to the system (or its components)?
- Who uses the system?
- Who uses which information of the system and for what purpose?
- What are the interactions between the system and its environment?

The enterprise concepts provide a framework for enterprise specifications and consist of basic concepts, which are necessary to represent an ODP system in the context of the organisation in which it operates. An enterprise model is defined in terms of:

- *Roles* that are played by the enterprise objects involved for the purpose of achieving an objective. For instance, the enterprise object ‘Dutch PTT Telecom’ plays the role of network operator to provide the mobile telephone infrastructure;
- The *Community* concept is used for grouping enterprise objects to meet an objective. The mobile telephone service in Figure 2.2 can be achieved by a combination of end-users, third party retailers offering mobile phones and subscriptions and a mobile network infrastructure;
- *Policies* are expressed in terms of permission, i.e. what can be done, or prohibition, i.e. what cannot be done, and obligation i.e. what must be done.

Policies set down rules on which actions of which objects are permitted or prohibited, and also which actions objects are obliged to be carried out;

- An important concept is that of a *contract*, linking enterprise objects that perform a certain role and expressing their mutual obligations. A contract can express the common goals and responsibilities, which distinguish the roles in a community. For example, the Dutch PTT Telecom in Figure 2.2 as mobile network operator, and its customers (e.g. Family Brown) have a contract that specifies their mutual obligations with respect to the mobile telephone service.

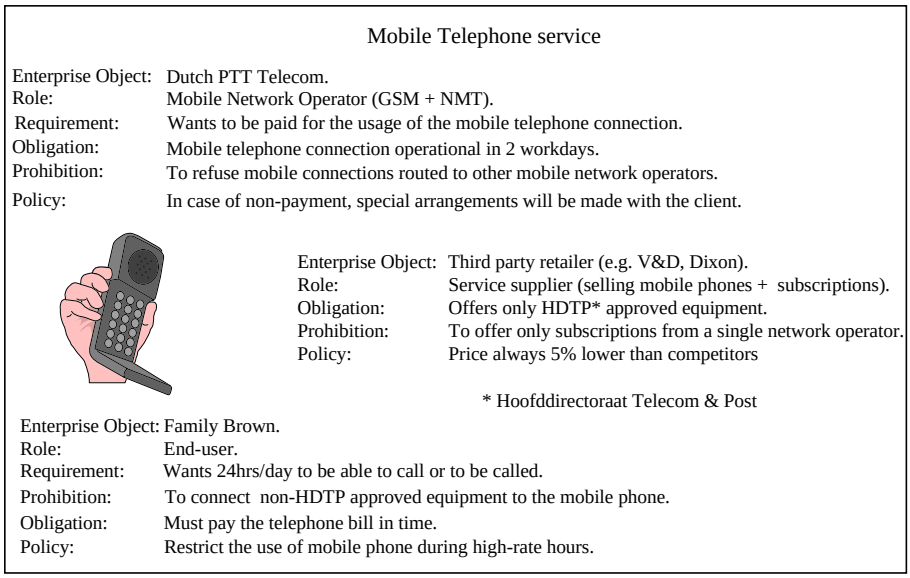



Figure 2.2: Example enterprise specification of a mobile telephone service

- A *federation* is a particular kind of community. It is defined as the property of combining systems from different administrative or technical domains to achieve a single objective. The evolution of distributed systems will result in the merging of existing, separately autonomously managed sub-systems to share information or support common commercial interests. The creation of federations and the expression of the rules to control federations forms an important part of the system specification in the enterprise viewpoint.

Where appropriate, an enterprise specification will also express aspects of ownership of resources and responsibility for payment for equipment/software and services.

### 2.3.2 Information language

The information language focuses on the information semantics and information processing activities in a system. Components of a distributed system must share a

common understanding of the information they exchange, or the system will not behave as expected. The information language defines concepts that enable the specification of the structure and semantics of information stored within, and manipulated by, an ODP system, independently of the way the information processing functions themselves are to be distributed. The information specifier describes information held by the ODP system in terms of information objects, and their relationships and behaviour. The information specification can be composed of the combination of the following statements [18]:

- What are the information objects of the system?
- What manipulations/processing can be performed on the information objects of the system?
- What is the relationship between information objects?
- What are the attributes of the information objects?
- What are the rules and constraints for information manipulation?
- What are the sources, sinks and information flows in the system?
- What semantics would be associated by a human with the information that is stored and exchanged between information objects?

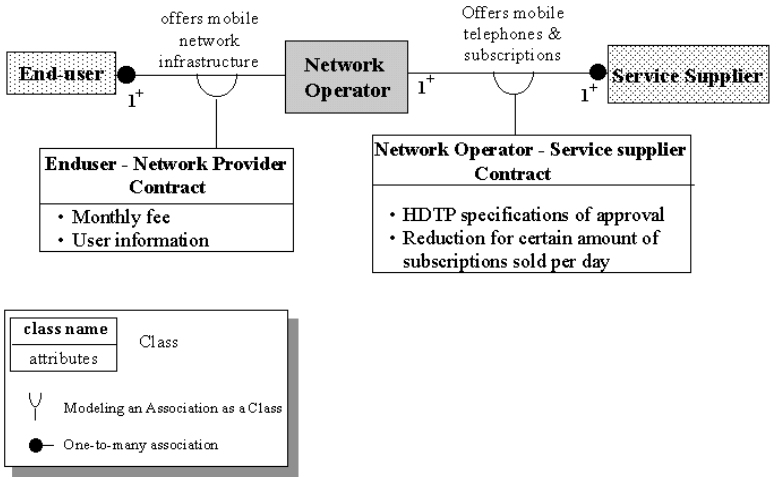



Figure 2.3: Example of a static schema using OMT graphical notation

The information specification consists of a set of related schemata. ODP-RM distinguishes three types of schemata: invariant schema, static schema and dynamic schema.

- An *invariant schema* expresses relationships between information objects, which must always be true, for all valid behaviour of the system. It restricts the state and structure of information objects at all times. An invariant schema for the mobile telephone service example (Figure 2.3) might specify that the user information

(i.e. name, address but also a user profile) between an end-user- network operator contract must be always up-to-date;

- A *static schema* expresses assertions, which must be true at a single point in time. A common use of static schemata is to specify the initial state of an object. For the mobile telephone service example the monthly fee agreed between end-user and network operator might be fixed at the beginning of the agreement but adapted when additional telephone services are added;
- A *dynamic schema* specifies how the information can evolve as the system operates. It describes the permitted changes in state and structure of an object subject to the constraints of an invariant schema. In addition to describing state changes, dynamic schemata can also create and delete information objects. For example, the telephone service requires a dynamic schema for calculating the usage of the mobile telephone by its customers. Another example is that a user might use 900 numbers unless stated in the end-user profile (expressed in the invariant schema).

An *information object template* in ODP consists of a combination of static schema, invariant schema and dynamic schema.

Integrity rules describe assertions about a schema. For example, the network operator has a record for each end-user in its telephone-usage-database, and every record in the telephone-usage-database is owned by one end-user (i.e. no empty records are allowed).

Part of an information specification is the description of relations that describe the associations between information objects. In Figure 2.3, the relation between the network operator and the service supplier information objects is characterised as ‘a service supplier may offer telephone equipment if it satisfies the HDTP rules’.

Various information specification notations are available, which model the properties of information in different ways. Emphasis may be placed on classification and reclassification of information types, or on the states and behaviour of information objects.

### 2.3.3 Computational language

In the computational viewpoint, a distributed application consists of a collection of computational objects. A computational object provides a set of services that can be used by other objects. An object offers a computational interface to enable other objects to access its service. This is the only means by which other objects can use that service.

The ODP computational language (or model) is used to describe the structure of a distributed application and specifies the interaction between computational objects thereby describing the service and behaviour of the distributed application. Using the computational modelling concepts one can specify a distributed application without

worrying about the details of the underlying distributed processing platform. The design principle of the computational model is to minimise the amount of details that the application programmer is required to know. A computational specification of a distributed application consists of the composition of computational objects (which represent application components) interacting by invocations at their interfaces. It identifies the activities that occur within the computational objects and the interactions that occur at their interfaces.

A computational specification addresses the following statements:

- What are the computational objects of the system and how are they structured?
- What are the interfaces of the computational objects?
- What operations can be invoked to and from the computational interfaces?
- What is the role of the computational interface?
- What behaviour is observable at the computational interfaces?
- What environment constraints are associated with the computational objects and associated interfaces?
- What interaction is possible between computational objects (interfaces)?

The ODP computational language provides templates for the specification of computational objects and interfaces. These templates enable the application designer to express the constraints on the interfaces and objects to be designed. The computational language also comprises concepts, rules and structures for the specification of an ODP system from the computational viewpoint.

Use of a computational language does not prevent the use of software designed for proprietary (non-open) systems in a distributed environment. It allows encapsulation of existing applications as components of a larger, distributed application. This permits an evolutionary approach, thereby protecting investments in existing software.

The following sections discuss the most important computational concepts, which are computational interface, object template and binding.

#### 2.3.3.1 Computational interfaces

Interfaces are the only places where an object can interact with another object to obtain services. A computational interface is characterised by a signature, a behaviour and an environmental contract. The signature describes the interface type, which can be either *signal*, *operation* or *stream* interface. The behaviour and environment contract are similar for the object template and described in Section 2.3.3.2.

### Signal interface

Signal interfaces are elementary interfaces. The signal interface initiates signals and might receive responding signals. A signal is an elementary atomic interaction. The signal interface signature describes the characteristics of each signal. Both operation and stream interfaces can be decomposed into signal interfaces. An example of a signal interface would be clock interface emitting a pulse once every second.

### Operation interface

The operation interface is an interface whose signature describes a set of operations. An operation is similar to a procedure and is invoked on designated interfaces. Operations reflect the client/server paradigm. An operation is an interaction between a *client object* and a *server object* to execute a function by the server and return the results to the client. There are two types of operation:

1. An *interrogation*, in which the server returns a response (a termination) to the client request (request-response style similar to Remote Procedure Calls (RPC)). The notion of termination represents results and exceptions as found in many both object-based and non-object-based programming languages;
2. An *announcement*, in which there is no response to the client request (request-only style).

### Stream interface

The stream interface describes behaviour, which consists of a single non-atomic action that persists throughout the lifetime of the interface. A stream interface may consist of a number of related flows, either in one single direction or in opposite directions. It can be characterised as time-based (isochronous) information such as audio or video. A flow is an abstraction of continuous sequences of data between interfaces. The stream interface signature contains the type of the flow and an indication of causality (i.e. direction of flow). Table 2.3 summarises the various interactions that are described by the computational model.

| Interaction | example | interface type | causality | signature |
|---|---|---|---|---|
| **Signals** | atomic action for measurement purpose in QoS observations | signal interface | **initiating**: initiates signals<br><br>**responding**: responds to signals | for each signal: name for the signal number, name, type of parameters causality |
| **Flows** | video sequence, audio samples | stream interface | **producer**: generates flows<br><br>**consumer**: consumes flows | for each flow: name of the flow number, name, type of parameters causality |
| **Operations** | a procedure | operation interface | **client**: invokes operations expects terminations<br><br>**server**: expects operations responds by one of the terminations | for each operation: name of invocation number, name, type of parameters if interrogation then for each possible termination: number, name, type of parameters causality |

Table 2.3: Object interactions and associated interface type

### Discussion on interface types

In many architectures (OMA, Microsoft OLE, ANSA), the application designer does not decompose the operation or stream interface into the elementary granularity of the signal interface. Signal interfaces have been introduced, and are used by CNET and Lancaster University to model interactions with so-called reactive objects [22]. The reactive object model is a formalism that merges the object-oriented approach with the assumption of a global time. This means that objects do not execute at their own rate but are instead driven by a logical common clock. In this model, operations can be invoked that are non-blocking signals that are processed immediately (thus a delay of zero). For this model signals are used, which are represented by the ODP signal interface. De-facto standards that become commonly accepted such as CORBA only consider less elementary interfaces, i.e. operation interfaces.

Operation interfaces have been implemented for all DPEs. The operations supported by an operation interface are expressed in an Interface Definition Language (IDL), which is used to generate software (i.e. stub routines) for the coding and wrapping of operation parameters. Operation interfaces have powerful characteristics such as unit of reuse, machine independent and programming language independent (see [51] for details). This explains their widespread support.

Stream interfaces have a time-based interaction mechanism that is needed for multimedia services. The main difference with discrete data interaction is that time-based interaction has a certain duration in time. QoS for operation interfaces focuses on reliability of the operations, whereas QoS for stream interfaces puts a priority on timeliness of the dataflows. The QoS characteristics, determined by the communication

and operating systems, determine the time-based interaction to a great extent. Another difference between operation and stream interfaces is that the direction of the information exchange is not predetermined for stream interfaces. In a time-based interaction, the exchange can have any direction: from one object to another, vice versa or even bi-directional. In contrast to operation interfaces, the request is always from the client to the server and the reply from the server to the client.

### 2.3.3.2 Computational Object template

A computational object template comprises of a set of interfaces, which the object can instantiate, a behaviour and an environment contract. Each interface is characterised by a signature, a behaviour and an environmental contract. The signature depends on the interface type, which can be operation, stream or signal.

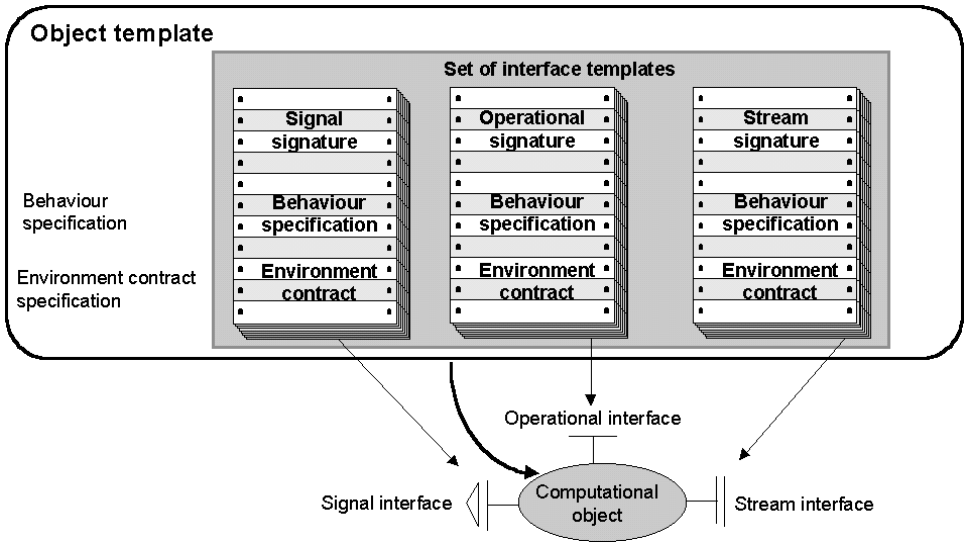



Figure 2.4: A computational representation of objects and interfaces

The behaviour specification of an object is defined as the sequencing constraints and concurrency constraints applicable to the object. It defines the overall behaviour of the object, which might in turn constrain the behaviour for each interface that is supported by the object. The *behaviour* in a computational interface template describes the allowed sequences of actions of the interface and focuses on the functional behaviour of the object.

The environment contract specification of the object template applies to the object as a whole, including its interfaces and focuses on the non-functional properties related to the object. Constraints, which are specified in the object environment contract, could be:

- The object can only be located in a certain domain (security constraint, location constraint);
- The object has a specified maximum probability of failure (reliability constraint).

Each interface specification contains an *environment contract*, which specifies a set of constraints on that interface and its environment. A particular notation for specifying the constraints is not prescribed by the computational language. In Chapter 5, a notation is proposed for the specification of an environment contract.

#### 2.3.3.3 Binding

Interactions between computational interfaces are only possible if a *binding* (i.e. a relation exists between the interfaces, using a communication path) has been established between them. The computational language specifies *explicit* binding actions for both operation and stream interfaces. For operation interfaces, it also specifies that binding can be *implicit.* In this case, no particular requirements or control will be expressed at the computational level on this binding. Implicit binding is only possible for operation interfaces since for stream interfaces there is a need to have control over the stream. Implicit binding is not explicitly modelled in the computational specification and automatically solved by the engineering mechanisms (see Section 2.3.4).

Where binding is explicit it is defined in terms of two kinds of binding actions: *primitive binding actions* and *compound binding actions*.

A primitive binding action allows the binding of two interfaces of the same or of different objects. It is initiated by one of the objects involved and has the effect of establishing at each interface the necessary information for interaction to take place. Thus the identity and (QoS) properties of the other interface are exchanged. A precondition for a successful primitive binding action is that the involved interfaces are of same type but have complementary causality and signature types (e.g. one is a producer and the other a consumer).

A compound binding action allows the binding of two or more interfaces of the same or different type by means of a binding object. A compound binding action permits the specification of multiway bindings and complex bindings. For instance, bindings between operation or stream interfaces of different types are allowed. The action can be initiated by one of the objects involved in the binding or by a separate (third party) object.

**Binding object**

The binding object is used to abstract from end-to-end connections and is responsible for compatibility checks between the involved interfaces. The behaviour of binding objects reflects the communication semantics they support. As with any other object, binding objects can be qualified by QoS declarations that further constrain their correct

behaviour (e.g. end-to-end communication delay or end-to-end delay-jitter at a recipient interface). The computational model does not restrict the types of binding objects, and allows various possible communication structures between objects. However, useful classes of binding objects may be standardised depending upon the classes of applications.

The following figure shows an example of the actions performed to instantiate a compound binding [21].

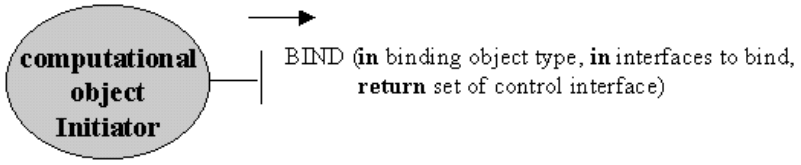



Figure 2.5: Step 1: Compound binding action example

The `bind()` operation has the effect of instantiating a computational object from a binding object template that supports the binding (Figure 2.6, the binding object).

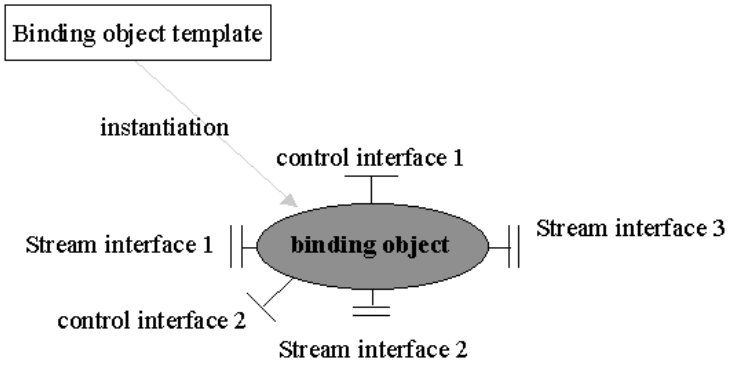



Figure 2.6: Step 2: Compound binding action example

The binding object instantiates an appropriate set of interfaces (Figure 2.6, stream interface 1..3) and uses primitive binding actions to bind them to the interfaces to be bound (Figure 2.7, set of primitive bindings). The binding object also instantiates a set of control interfaces (Figure 2.6, control interface 1..2) through which the binding can be controlled. The binding object returns the control interface references to the initiating object. If the binding action fails, information about the error will be returned as a result.

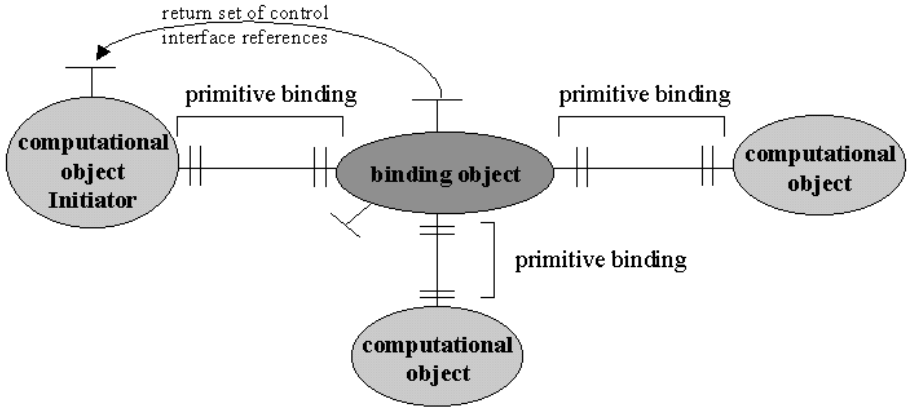



Figure 2.7: Step 3: Compound binding action example

### 2.3.4 Engineering language

The engineering language focuses on the way object interaction is achieved and the resources needed to do so. Thus the computational viewpoint is concerned with *when* and *why* objects interact, while the engineering viewpoint is concerned with *how* they interact. In the engineering language, the main concern is support of the individual interactions between computational objects. It is here that one of the most direct links between viewpoints is found. Computational objects are visible in the engineering viewpoint as *basic engineering objects* (BEOs) and computational bindings are visible as *channels* or local bindings.

This section discusses basic concepts of the engineering language: the node components (Section 2.3.4.1) the way the node interacts (Section 2.3.4.2), and the functions provided at the engineering level (Section 2.3.4.3).

#### 2.3.4.1 Clusters, capsule, nodes and basic engineering objects

The engineering language deals with basic engineering objects (BEOs) and various other engineering objects that support the BEOs. It relates the BEOs to available system resources. Objects are physically grouped into *nodes* which associates them with processing resources. Nodes can be regarded as autonomously managed computing systems. A node can be for instance a PC or a telecommunication switch. A node is considered as a strongly integrated view of resources, which can be considered by the system designer as a whole.

A node is under the control of a *nucleus* (Figure 2.8), which is responsible for initialisation, creation of object groups, provisioning of communications facilities. The nucleus also provides basic services such as timing and generation of unique identifiers.

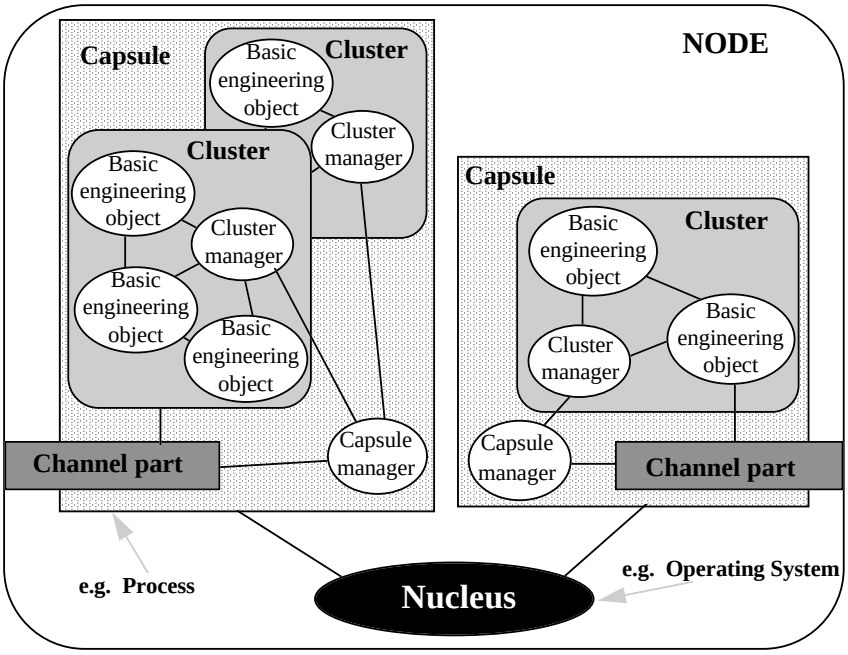



Figure 2.8: Engineering concepts

A node can contain a number of *capsules*. A capsule owns storage and a share of the node's processing resources. It can be compared to a UNIX process with its own address space. A capsule is the unit of protection and is generally the smallest unit of independent failure supported by the operating system. There is a special object, called the *capsule manager*, associated with each capsule and a capsule is controlled by interactions with this manager.

A capsule can contain many objects. Grouping of objects into capsules reduces the cost of object interaction. Communication between processes is slower because of the various checks, which need to be performed. Resources within a capsule will be controlled by a specific operating system. The smallest grouping of objects is a cluster. Objects in a cluster are grouped together in order to reduce the overhead to manage them (e.g. instantiate, delete the objects etc.).

The set of objects in a cluster can be checkpointed, transferred to persistent storage, reactivated or moved to another node altogether. Manipulation of complete clusters as a single operation opens the way to the management of fine-grain object-based systems. Communication between objects in a cluster can be optimised, since the objects are created together and are expected to stay together. Interaction within a cluster might therefore be supported by a simple local operation. Clusters are controlled and actions on them initiated by interaction with an associated *cluster manager* object.

### Instantiation of engineering objects

Engineering objects need to be created on a node before other objects can use these objects. The process of creating objects on a node is not prescribed by ODP-RM since the way in which objects are instantiated depends on the implementation of a specific node. However, to obtain understanding about object creation in a distributed environment, a scenario will be described that is closely aligned with the object creation process in OMG CORBA [20].

From a client's point of view, objects are created using the function `new_object()`. This function uses the class name and attributes described in an environment contract specification as input, and returns an interface reference. Figure 2.9 shows the process of instantiating an object using the factory function.

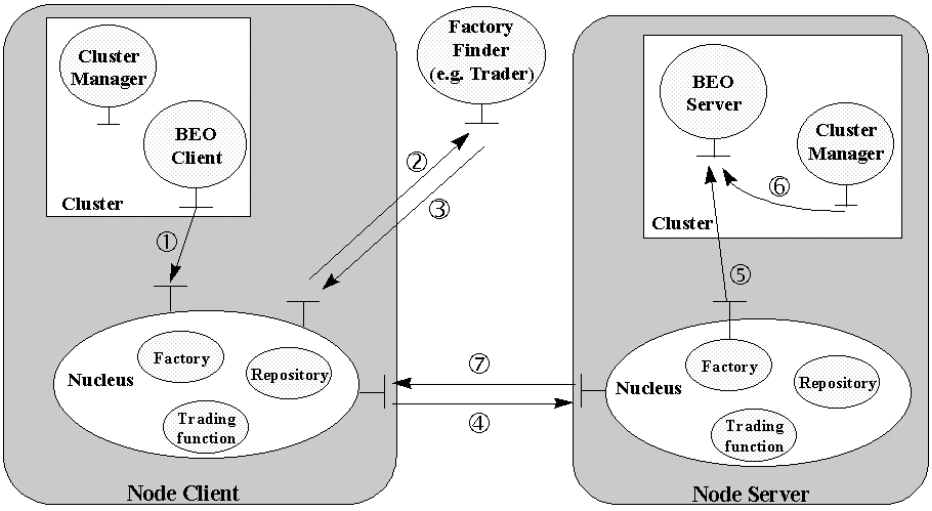



Figure 2.9: Process of object instantiation

The client requests the trading function in its Node for a reference of a server satisfying the requested class and capability list expressed in an environment contract (Figure 2.9, ①). The trading function will contact a factory finder (②) that can be a kind of a trader that matches the request to suitable factories that satisfy the request. The Factory finder will respond with a possible list of references to factories that match the request (③). The client node sends a `create-object()` operation to the nucleus where the object has to be created (④). The factory in the server nucleus will instantiate the server object with the requested capability list (⑤). When the server object is no longer needed an operation `remove-object()` is invoked which removes the object (⑥). Clients using the server object will be notified (⑦).

#### 2.3.4.2 Engineering binding (channel)

This section presents concepts and rules related to the binding of engineering objects located in different nodes. It first presents the components of the channel, then it describes the structure of the interface reference passed between the node involved and finally it shows how the channel is established.

### Channel objects

There are two types of engineering binding between objects. The first type of binding is where objects are grouped in one cluster. Binding between these objects is called local binding and is resolved by systems-specific mechanisms. The second type of binding is between objects that are **not** grouped in one cluster. To bind these objects a *channel* is needed.

A channel is a configuration of stub, binder, protocol and interceptor objects that interconnect a set of engineering objects (Figure 2.10). *Stubs* are concerned with the (de)coding of the information conveyed in an interaction. *Binders* are concerned with maintaining the association between the set of basic engineering objects linked by the channel. *Protocol objects* manage the actual communication.

Stubs interact directly with the basic engineering objects they support and perform functions such as marshalling and unmarshalling of parameters or logging of information about the interaction being performed. Thus the stubs need access to information about the type of the interaction, or, more generally, the type of interface being supported. This distinguishes them from binders and protocol objects, which transfer complete messages without concern for their internal structure. Depending on the design of the system, a stub can be directly associated with a particular basic engineering object or it can be shared between a number of such objects as shown in Figure 2.10. Sharing will generally imply the need to transfer additional information to identify and thus to distinguish between the objects being supported.

Binders address many of the problems associated with distribution. They are responsible for maintaining the end-to-end integrity of the channel and deal with object failures. Consequently binders have to handle configuration and communication changes. A binder has to establish the binding when the channel is created, and has to keep track of endpoints if objects move, are replaced or fail. The binder object is involved in the process of object relocation. The binder object is involved in the provision of many of the distribution transparencies described in Table 2.1.

The protocol object insures that BEOs can interact (remotely) with each other. Protocol objects are needed if the computational objects that have to be bound are located in different nodes. A protocol object also provides access to supporting services, such as directory services for translating addresses.

The stub, binder or protocol object itself may need to communicate with other parts of the system, in order to obtain the information it needs to perform its task, or to supply management information to other objects. Such communication may itself need the various distribution transparencies, and so the communication from these objects to elsewhere is by means of a channel. From this point of view, objects within one channel can play the role of basic engineering objects in another channel. Similarly, any of these objects can support control interfaces, via which they can be managed. For example, a protocol object can provide a control interface through which the target quality of service for the channel can be adjusted.

In cases where the channel crosses a certain technical or organisational boundary, there can be a need for additional checks or transformations to match the requirements on the two sides. These functions are performed by *interceptors* that are part of the channel. They may need to perform format or protocol conversion, or may provide accounting or perform access control checks. An interceptor can be built up from protocol objects, binders and stubs, depending on the nature of the task it has to do.

Channels with multiple endpoints can be defined, to support various forms of group communication or multicast. In multi-endpoint channels, the binders are responsible for co-ordinating communication, but either binders or protocol objects, depending on the technology available can provide the multicast mechanisms. Multi-endpoint channels are used to support replication transparency.

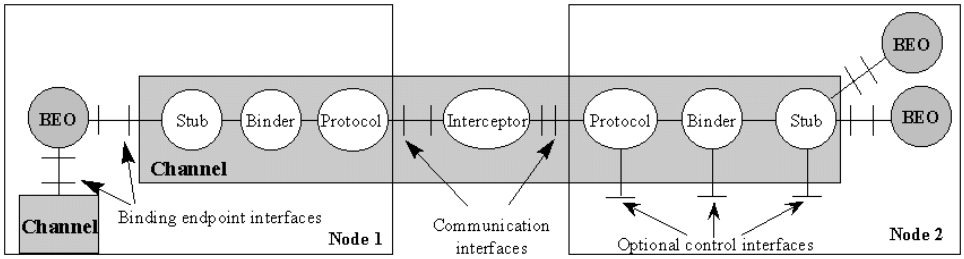



Figure 2.10: An example of a channel

A *communication interface* is the interface of a protocol object, which can be bound to an interface of either an interceptor object or another protocol object. A *binding endpoint identifier* is used by a BEO to select one of the bindings in which it is involved, for the purpose of interaction.

### Interface references

In order to establish a binding between engineering objects, it is necessary to be able to identify and describe the interfaces of those objects. An interface reference is used to identify and describe an engineering interface in sufficient detail to enable that interface to be bound. An interface reference can be passed through one or more interactions

between engineering objects. It enables any object, which receives that interface reference to initiate a binding to that interface without any additional information.
The type of an engineering interface must describe the computational interface type that it implements, in addition to the channel configuration of stubs, binders, protocol objects and interceptors needed to support it. When engineering interfaces with compatible computational interface types are bound together, the binding establishment must negotiate the exact configuration of the channel needed to support all the interfaces it encompasses. It is possible to optimise this configuration. For example, if the interfaces support a common representation of data, then there will be no need for the channel to be configured with objects that perform conversion between the data representations. Although it will still be necessary to marshal the arguments or results into a message.

Interface references unambiguously identify an interface. The interface reference conveys complex information, which should be interpretable throughout the ODP system. In an ODP system it is assumed that the nucleus is responsible to create an unambiguous interface reference. Given such a reference, it is possible to discover the type of the interface, and to determine a communication address at which the binding can be initiated. The interface reference also contains other information about the expected behaviour of stubs, binder and protocol objects within the channel.

If the content of an interface reference becomes too large for convenient storage or transmission, then all or part of the content can be held in a directory. The content is replaced in the interface reference by a small key, which can be used to access that content from the repository when required. The objects passing such interface references must have the means of determining the repository at which the content is held. For example, in certain systems, this is a well-known interface.

The generic structure of an interface reference is not prescribed in the current ODP-RM standard and is scheduled for standardisation in ‘96/97. ODP-RM does not prescribe whether the interface reference must physically contain the information described there, or whether it is simply a key for accessing this information through interaction with other objects (e.g. a name to be resolved by a name server).

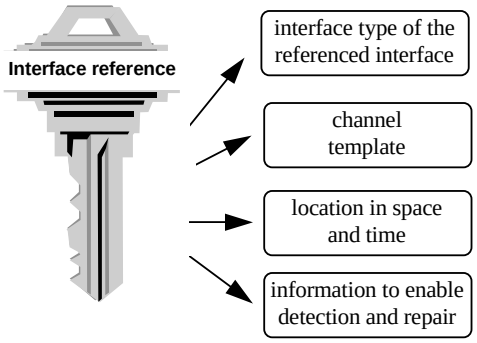



Figure 2.11: Elements described in an interface reference

In [23] an initial attempt is made to define the syntax of an interface reference, which is represented in Backus Naur Form (BNF) in Table 2.4 [24].

| | |
|---|---|
| <interf-ref> | ::= <null> \| <direct reference> \| <non-interpreted-reference> |
| <direct-reference> | ::= <interf-type> <channel-class> <location-info> <relocation-info> <additional-info> |
| <non-interpreted-reference> | ::= <interpreter-reference> <opaque-info> |

Table 2.4: BNF description of interface reference structure

Focusing on the `direct reference` part, the `channel-class` provides information about the stub, binder and protocol objects used in the channel local to the referenced interface. This information is needed to configure the remote end of the channel using equivalent stub, binder and protocol objects. The channel class also identifies the supporting objects required for the channel (e.g. interceptor).

The `location-info` provides the necessary information for the binding to find the location of the interface at creation time. The information includes network and node-specific addressing. An interface reference may include a set of different pieces of addressing information, corresponding to different access paths. Location information will be context sensitive, because of this association with possible access paths. The format of the location information will depend on the channel class in the interface reference.

The `relocation-info` identifies a relocator object that manages a repository of locations for interfaces, and this information is used when an interface changes location. The `additional-info` provides a place-holder for functions not directly related to binding, e.g. interface reference tracking

The `non-interpreted-reference` is used in systems crossing different interface reference domains where different naming policies apply (e.g. CORBA object reference [25] versus ANSA interface reference [26]). The `interpreter-reference` identifies an object able to replace the `opaque-info` part of a `non-interpreted-reference` with a `direct reference`. For instance, a translator object, which is able to convert a CORBA object reference to an ANSA interface reference.

It should be noted that in this definition of interface reference structure explicit support for QoS specification is not included. In Chapter 5 extensions are proposed to the interface reference structure that also includes placeholders for the specification of QoS.

### Channel establishment

Channel establishment is accomplished through the interaction of the nucleus objects. Therefore, when a nucleus creates an interface reference, it must supply sufficient instructions on how to be contacted in order to establish a binding to the referenced interface. In engineering language terms, the nucleus must identify the *communication*

*interface*(s) at which the nucleus-nucleus interaction must occur. The identification of a communication interface might require information about the communication protocols supported at that communication interface.

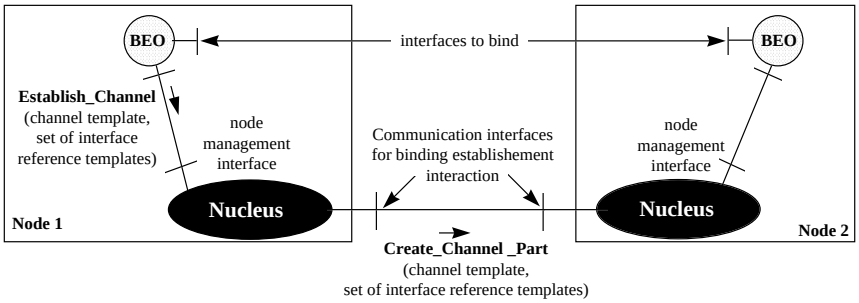



Figure 2.12: Step 1: Channel establishment example

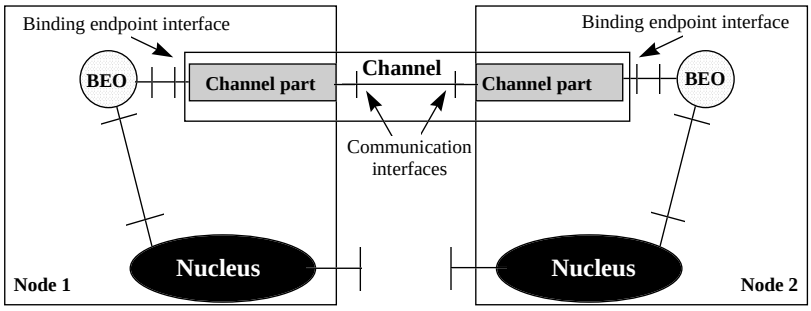



Figure 2.13: Step 2: Channel establishment example

Typically, the nucleus will nominate one of its own communication interfaces for binding-establishment interactions, but a more general approach is also permitted. The communication interface (and its communication protocol) for nucleus to nucleus communication which is used for the binding-establishment does not imply anything about the communication interface (and communication protocol) that will be used to support the binding between BEOs once it is created. This can be compared to outband signalling where the signalling channel and data channel are separated and can use different protocols.

### 2.3.4.3 Functions

ODP-RM defines a number of common functions at engineering level which are fundamental for the construction of ODP conform systems. Several of these functions are already included in the engineering language such as the stub, binder and protocol objects that are needed to support the engineering infrastructure. The second category of functions is introduced to provide support various transparencies as defined in Chapter 2. Table 2.5 shows the collection of functions as defined in ODP-RM which are classified in four groups (first column).

| Function category | Function name | Role |
|---|---|---|
| Management | node manager | controls processing storage and communication functions within a node |
| | capsule manager | instantiates, checkpoints and deletes all the clusters in a capsule and deletes capsules |
| | cluster manager | checkpoints, recovers, migrates, deactivates or deletes clusters |
| | object manager | checkpoints and deletes objects |
| | migration | co-ordinates the migration of a cluster from one capsule to another |
| | checkpoint recovery | co-ordinates the checkpointing and recovery of failed clusters |
| | deactivation/reactivating | co-ordinates the deactivation and reactivating of clusters |
| | event notification | records and makes available event histories |
| Co-ordination | transaction | co-ordinates and controls a set of transactions to achieve a specified level of visibility, recoverability and permanence |
| | ACID transaction | special case of transaction |
| | replication | ensure a group appears to other objects as if they were a single object |
| | group | co-ordinates the interactions of objects in a multi-party binding |
| | engineering interface reference tracking | monitors the transfer of engineering interface references between engineering objects in different clusters |
| | relocator | manages a repository of locations for interfaces |
| | storage | stores data |
| Repository | type repository | manages a repository of type specifications and type relationships |
| | trading | mediates advertisement and discovery of interfaces |
| | information organisation function | manages a repository of information |
| | access control | prevents unauthorised interaction with an object |
| | security audit | provides monitoring and collection of information about security-related actions |
| | authentication | provides assurance of the claimed identity of an object |
| Security | integrity | detects and/or prevents the unauthorised creation, alteration or deletion of data |
| | non repudiation | prevents the denial by one object of having participated in all or part of the interaction |
| | key manager | provides facilities for the management of cryptographic keys |
| | confidentiality | prevents the unauthorised disclosure of information |

Table 2.5: ODP-RM functions

ODP-RM identifies a number of functions that meet the requirements for a wide range of applications (e.g. the *trading* function or the transaction function). ODP-RM

provides a framework for directly implementable standards for these generic functions, which permits a number of different approaches to their realisation.

### 2.3.5 Technology Language

The technology specification describes the implementation of the ODP system in terms of a configuration of objects representing the hardware and software components. The technology specification is constrained by cost and availability of technologies (i.e. hardware and software products) that satisfy this specification and may conform to implementable standards. Thus, the technology viewpoint provides a link between the set of viewpoint specifications and the real implementation. It also describes additional information needed for the implementation and testing of the system by selecting standard solutions for basic components and communication mechanisms.

## 2.4 Discussion

In 1995, ODP-RM became an international standard (IS) following a standardisation process of eight years. Initially, the ideas of ODP were rather new and advanced, especially for the telecommunications industry. During the course of the standardisation process, ODP has influenced many other de-jure standards (e.g. ODMA), industrial consortia (e.g. OMG, TINA-C) and international projects (e.g. Cassiopeia [107], ROSA, UMTS, ReTINA). The TINA consortium is one of the first consortia that use the ideas of ODP-RM extensively. Despite the acceptance of ODP-RM, several weak points of the ODP-RM model have been identified that could be improved.

**Viewpoint languages**

The core of the ODP architecture is found in the computational and engineering viewpoints. Here the concepts and structures for distributed applications and the infrastructure they need are described. The author observed that ODP's computational and engineering model and the concept of viewpoints are widely accepted outside the ODP community. Other ODP viewpoints are less used or interpreted differently. This is mainly caused by the generality of the ODP enterprise, information and technology viewpoint languages, which allow quite some freedom for the application designer. In particular, the technology viewpoint language is empty. This can be explained by the fact that ODP-RM serves as a general framework and does not want to include technology details since this would outdate a framework rather quickly (technology evolves so fast that flexibility is required to incorporate new technologies).

The immaturity of the enterprise and information viewpoint languages is due to historical reasons. When ODP standardisation started, the focus of certain experts in the group was on the enterprise/information viewpoints whereas the majority was interested in the computational and engineering viewpoints. Partially due to misunderstanding and different interests of the two camps, no consensus could be reached and slow progression of the standard was the result. In 1993 the problem was solved when a new

working group 'Conceptual Schema' was initiated to focus on the enterprise/information viewpoints for ODP. However, the scope of their work differs from the original ideas behind the enterprise and information languages. It is therefore unlikely that the results of the conceptual schema group will be used for these two viewpoints. The ODP group is aware of this problem and a new work item was started in 1996 to develop the ODP enterprise language using input from the OMG Business Domain Task Force. Unfortunately, the business objects, as defined by OMG [27], are merely a computational extension of grouping several objects into business objects (e.g. an IDL specification of the business object is required). Again this differs from the ideas behind the enterprise language, which should focus on analysis of the problem domain.

This shows once again that the scope and purpose of the ODP enterprise language are not clearly defined. From the author's experience it is known that the enterprise viewpoint has often been confused or used together with the information viewpoint. This could be explained by the fact that application designers use object oriented analysis & design (OOA&D) methods (e.g. OMT, BOOCH, FUSION, and Jacobson) as a practical methodology for the enterprise and information viewpoint languages. The OOA&D methods encompass both enterprise and information viewpoints (others even include the computational viewpoint) without a clear distinction. It would have been more appropriate to reduce the five viewpoints to four by combining the enterprise and information viewpoint into one 'Analysis' viewpoint. This viewpoint should then use the concepts and rules defined by well-accepted OOA&D methods.

A further aspect is the immaturity of the ODP-RM standard regarding the relationship between different viewpoint specifications. Although this is necessary to achieve consistent specifications, very few rules are described in ODP-RM to achieve this. In the current literature this issue is also not well developed and the author believes that future ODP activities should focus on this issue.

### ODP functions

The engineering functions in ODP-RM are described in general terms. For most ODP functions the author believes that the ITU-T/ISO ODP standardisation group will be too late to influence consortia like TINA-C and OMG. In fact the process now works the other way around: on-going work on the ODP functions corresponds more to the standardisation of existing concepts and mechanisms of the functions specified in various consortia. The ITU-T/ISO ODP standardisation group is adopting, in particular, OMG services to become the standards of ODP functions. In fact, to fast track the OMG standards and to enforce synergy with de-facto standards, reduces the delays to reach IS status for ODP functions. One exception is the trader, which is the most developed function in the ODP-RM. Since OMG has issued a trader RFP, and the ODP trader specification has reached the DIS status (including a computational specification that uses OMG IDL), it is put forward to become an OMG standard.

The question arises then to which extent the ODP functions together provide a consistent 'implementation' of an ODP conform system when most of the functions are implemented by OMG standards. Historically, many of the ODP functions were derived from ANSA and their product ANSAware. However, ANSAware is not widely accepted by the industry and regarded as an experimental platform with limited influence. With the new de-facto standard CORBA becoming now broadly accepted by the industry, the author believes that the ODP functions are less important. The ODP-RM functions just indicate what kind of functionality should be present in an open distributed system.

### Standardisation process

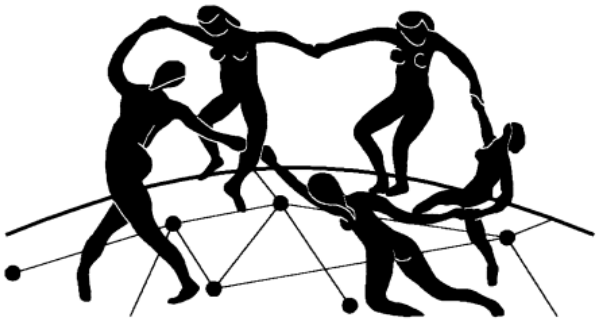
The author participated in the ODP standardisation process from 1991 onwards and contributed to the discussions that were needed to shape the ODP reference model. In particular, contributions were made in the multimedia domain and on how to model distributed applications using concepts and rules defined by ODP-RM. Results of these contributions can be found in ODP Part 1 where examples from the author are included. Since ODP Part 2 and Part 3 are intended to be formal and abstract many people consider Part 1 very useful to actually understand the principles behind ODP-RM. It is obvious that ODP Part 3 is the most used part. On the contrary, ODP Part 4 is not very popular and probably only used in academic circles to formalise ODP concepts. The use of the formal language Z is not widespread outside the academic community and most designers prefer to use more practical methods for information modelling such as OMT.

The ITU-T/ISO ODP standardisation group is now focusing on the standardisation of particular components of ODP, such as, the repository function or the structure of interface references. The acceptance of these specialised components depends heavily on the alliance with de-facto standards groups.

The high speed of de-facto standardisation, and the impact of these standards because implementations of these standards are available, render the position and influence of the ODP standards (and in general de-jure standards) rather weak for the near future. This trend is also noticeable when considering the attendance and contributions to de-jure standards meetings. Far less people follow and contribute to ODP standardisation than was the case several years ago. This is partially caused by the completion of the most important parts of the ODP standard but to a large extent due to the fact that experts now prefer to participate in de-facto standardisation. This puts the future of the ODP standards in danger.

Nonetheless, the author believes that ODP-RM has already reached its objective by promoting object orientation in the telecommunications world, and separation principles (the ODP viewpoints) to tackle the design of complex distributed application. Additionally, ODP-RM provides a common set of concepts and terminology, which serves as a basis for the exchange of ideas between experts in the area of distributed processing. It is for this purpose that the standard is used in this thesis as a basis but will

refine it towards multimedia, and use the viewpoints where appropriate to distinguish between the various abstraction levels.



# 3 Industrial Consortia for Distributed Processing

*This chapter gives an overview of the different consortia[2] active in the area of distributed processing. The objective of this chapter is to give an impression of the forces at play and to highlight their scope and mutual influence. The consortia are compared on their approach towards object orientation, separation principles used and multimedia concepts incorporated in their DPE specifications. The need for a general framework is identified and the role of the ODP standardisation process to combine the architectures of both de-jure and de-facto standards is discussed. Hybrid standardisation should lead to the provision of a long lasting architecture using components provided by the various consortia.*

## 3.1 Introduction

Initiatives undertaken in the past resulted in several (often incompatible) solutions for the DPE. The rapid pace of development in the computer industry has resulted in the moving away of standards development from traditional standardisation bodies, such as ITU-T and ISO, towards industrial consortia such as TINA-C [71]. Figure 3.1 shows several consortia involved in information and communication technology standardisation, each addressing certain parts of it. For example: DAVIC defines standards for set top boxes used in the home environment, the TINA consortium [71]

2 The term consortia is used to indicate groups, organisations or individual companies that define and develop standards or products for the open distributed processing environment.

creates an overall (network) architecture suitable for the telecommunications industry and OMG defines open distributed computing architectures based on object orientation.

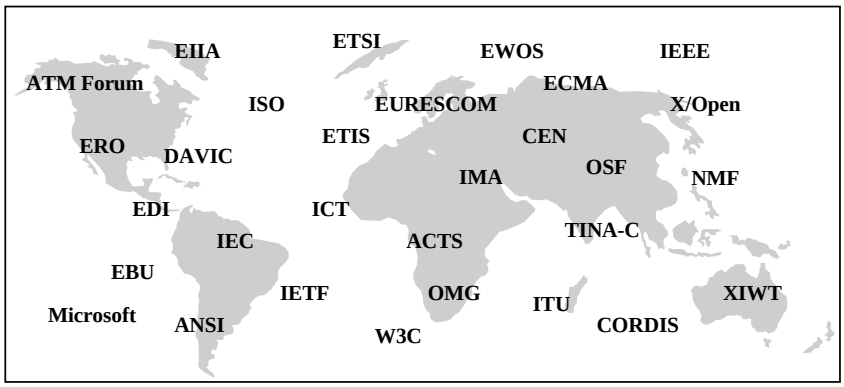



Figure 3.1: Several consortia and standards bodies[3] related to standardisation of information and communication technologies

In this fast evolving playing field the major players are identified, which exert influence on distributed computing during the last ten years. In a survey, the author encountered many organisations that deal with distributed computing, ranging from telecommunication standards for interconnecting switches from different vendors (e.g. ATM Forum), to proposals dealing with the development and introduction of (distributed) applications in business environments. In this thesis, the focus is on those organisations related to distributed processing environments that have an open character, or are powerful enough (e.g. Microsoft) to put forward a proprietary solution.

The objective of this chapter is to provide an overall picture of the forces at play and to highlight the scope of each consortia and their mutual interrelationships. Consortia discussed in this chapter often have different focuses. To enable comparison, several criteria are identified that are of importance for the remaining chapters. These criteria are: use of object orientation, separation principles used to design complex distributed systems, and availability of support for multimedia applications.

It is outside the scope of this chapter to provide technical details of the standards developed by each consortium (references to documents are included that elaborate on those consortia in detail). Several architectures are selected and analysed in more detail in Chapter 4. Additionally, the need for a general framework (ODP-RM), which can combine the various de-facto and de-jure architectures is identified.

3 For the full names of these consortia consult the abbreviation list at the end of this thesis.

## 3.2 The players

This section presents several industrial consortia. The consortia are selected based on the fact that they are fairly well supported by both the telecommunication and computing industry in the area of distributed processing and provide an open solution. Many consortia have the strategy to adopt other de-facto or de-jure standards (e.g. SPIRIT, OMG, OSF-DCE) using request for proposals (RFPs) to complete their specifications. In particular, OMG is well known for its RFPs, which enables the adoption of technology for its conceptual specifications in a short time frame. Other consortia, such as TINA-C intend to have a long term architecture and define more generic concepts and rules to specify distributed environments. The OSCA architecture developed by Bellcore was selected for its interesting separation principles. The specifications of IMA, DAVIC and MiTV support multimedia services of the ones studied. Others incorporate several multimedia features in their specifications (ODP-RM, TINA) or are in the process of incorporating them (OMG).

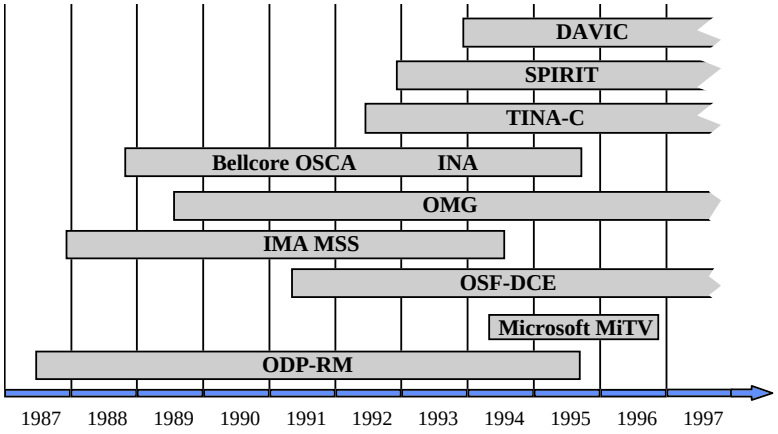



Figure 3.2: Consortia and standards related in time

Figure 3.2 shows the players investigated. They are positioned in time indicating when they started to develop their architectures. A short description of each of them is given below.

### Digital Audiovisual Council (DAVIC)

The DAVIC council was established in January 1994 and is an international non-profit organisation [37][38]. In September 1995, DAVIC consisted of 200 corporations representing businesses that have a strong relationship with digital audio-visual applications and services. The purpose of DAVIC is to promote emerging digital audio-visual applications and services. These services are broadcast and interactive services targeted for the home environment. Cost plays an important role and DAVIC promotes the idea of using open interfaces and protocols to achieve maximum interoperability

across countries and applications. This approach should ensure reasonable prices through the possibility of mass production of set top boxes. A set top box is a device normally located between the (cable) plug and television and is able to interpret incoming control signals and forward TV-signals it to the television set. Standardisation of these set top boxes is important since incompatible set top boxes on the market will prevent a wide spread usage of set top box based services offered by service providers. DAVIC's approach to standardisation is to make use of available specifications such as interface, protocol or architecture specifications as much as possible. Where necessary, it enhances existing standards and if needed, develops new standards. DAVIC has a close liaison and collaboration with several international standardisation groups (e.g. MHEG [80][146] and MPEG). Where applicable, DAVIC specifications are made available to the appropriate standards bodies.

An overview of the DAVIC architecture is given in Section 4.4.

### Service Providers' Integrated Requirements for Information Technology (SPIRIT)

SPIRIT is a joint effort of international telecommunications service providers, information technology suppliers and independent software vendors. The objective of SPIRIT is to provide a set of specifications that can be used in each company, which purchases software components for general purpose computing platforms. These specifications are driven by the requirements of the telecommunications industry [45]. SPIRIT is a team within the Network Management Forum. SPIRIT published its first set of specifications in 1993. The latest set of SPIRIT specifications was published second quarter of 1995. The SPIRIT specifications are based on widely accepted industry standards. SPIRIT expects that participating companies use the SPIRIT specifications as a basis for their own software procurement starting in the near term (within 1-2 years of publication).

The architecture of SPIRIT describes a generic software platform. It identifies the generic services to be present on a platform (i.e. a single standalone machine). For each of these generic services, SPIRIT provides a list of standards (de-facto and de-jure) that can be used to implement the generic services. These services are:

- *Operating system services:* that manage the fundamental physical and processing resources of a machine;
- *Management services*: that address systems management and network management standards and profiles;
- *Presentation services:* that act as the mediator between the system and human interface devices (display, keyboard, mouse, etc.);
- *Data management services*: that manage persistent storage;
- *Transaction services*: that co-ordinate resources in order to maintain transactional integrity over those data resources;

- *Communication services:* that define the protocols used in the distributed environment. The set of protocols ensure application interoperability in a SPIRIT compliant environment;
- *Distributed services:* that facilitate co-operative processing between two platforms;
- *Security services:* that enforce security policies on data and processing on the platform, and data objects exchanged between platforms.

### Interactive Multimedia Association (IMA)

IMA is an international association representing many companies dealing with multimedia [32][33][34]. IMA was established in 1988 and has more than 250 corporate, institutional and individual members. IMA defines a set of systems services that can be used by multimedia application developers in a heterogeneous computing environment. The IMA specifications date from September 1994 and are considered to be stable since no new releases are foreseen. IMA defines a set of Recommended Practices (RP) to facilitate cross-platform compatibility of multimedia data and applications. An important RP is the *Multimedia Systems Services (MSS)* specification.

The primary goal of the MSS specification is to provide an infrastructure by defining several generic objects for building multimedia-computing platforms that support interactive multimedia applications dealing with synchronised time-based media. It provides abstraction mechanisms that make it possible for applications to deal with the problems of distributed multimedia computing. The abstractions make it possible for applications to deal with media devices without regard to specific characteristics of the platform or attached devices to the network(s) connecting the platforms and devices. To support interaction between objects, MSS depends upon an underlying object infrastructure as defined by OMG's CORBA. For the exchange of time-based data, MSS defines its own media stream protocol (although not further developed by IMA). The services described by MSS can be regarded as a framework of system software components that are located between the local operating system and specific applications. The object-oriented approach is applied to specify the multimedia objects in MSS. The interface specifications of MSS objects inherit a subset of OMG's object capabilities such as the OMG life cycle operations (e.g. create, copy and delete operations). The MSS is intended to address a broad range of application needs. It extends the multimedia capabilities of stand-alone computers that are useable both locally and remotely. MSS gives applications the possibility to handle live data remotely. The MSS specification is discussed in Section 4.2.

### Open Software Foundation[4]

OSF is an international organisation founded in 1988. OSF developed several products that changed the computer industry into a more open systems industry [47][48][49].

4 The Open Software Foundation is nowadays called Open Group due to a merge with X/OPEN.

Examples of OSF products are OSF/Motif for UNIX GUIs and OSF/1 as an approach to open system operating systems. OSF's mission is to develop open environment software systems not tied to proprietary architectures or vendor products. By means of Request for Technology, OSF solicits technologies from the industry.

OSF's DCE was introduced mid 1991 as the technology that solves the problem encountered by computer vendors when the integration between UNIX systems and PCs became important. OSF-DCE allows for interoperability and distributed computing based on heterogeneous platforms and network technologies. The OSF-DCE specifications consist of two sets of services:

- *Fundamental distributed services* supplying basic software modules to assist application developers in creating end-user services for the DCE (e.g. Remote Procedure Call, Directory service, Time Service, Security Service and threads Service);
- *Data sharing services* supplying distributed file services to end-users such as distributed file system and diskless support services.

The main success of DCE is its RPC mechanism and its security and directory service. Microsoft uses the DCE RPC implementation for their Component Object Model (COM) but altered the DCE RPC specification so that is no longer an open standard as originally conceived by OSF.

### Object Management Group (OMG)

OMG is an international industry consortium comprising more than 500 companies in 1996, with every significant computer vendor represented [20]. OMG was founded in 1989 and is a non-profit corporation. OMG promotes the theory and practice of object-oriented software development. Its goal is to provide a common (object oriented) architectural framework over heterogeneous hardware and operating systems, for the intercommunication of application objects. OMG does not develop specifications itself but instead relies on (existing) technology offered by member companies. The OMG solicit specifications of components to fit in the overall Object Management Architecture through Request for Proposals (RFPs) on different areas of object technology. OMG forces that specifications proposed by member companies are complemented by an implementation proving the correctness of the proposed specification. Once a specification is accepted by OMG, any vendor that claims conformance to it can enter the market with alternative implementations. This process ensures that:

- OMG specifications are not just paper proposals but do actually work and,
- Several (competitive) implementations are available.

OMG has developed a reference architecture called Object Management Architecture (OMA). OMA defines the terminology for objects and the various facilities needed for distributed object-oriented computing. Most important part of OMA is the Common

Object Request Broker Architecture (CORBA). This is a general object communication architecture to support the development of distributed object oriented applications. CORBA provides the mechanisms necessary to identify, locate, access and manipulate objects in a distributed heterogeneous environment. The CORBA architecture consists of four components:

- *Object Request Broker (ORB):* is the basis of CORBA and allows remote operation invocation between distributed objects in the system. The ORB is an implementation of a distributed processing environment and allows the transparent communication between different platforms;
- *Object Services:* are fundamental object services, such as, the life cycle service for the creation and destruction of objects; naming service to locate an object in a distributed environment or event notification service to allow objects to notify interesting occurrences to each other. These object services must be provided by each CORBA compliant platform;
- *Common Facilities:* are high-level object services such as printing and error reporting services. These services are the basis for the development of application services;
- *Application services:* are business objects that are company specific services built on top of object services and common facilities.

### Bellcore

Bellcore was founded in 1984 and contributed to the development and evolution of voice oriented telephone services toward multimedia information networking. Bellcore contributed to the development of ISDN, Intelligent Networks and enabling technologies for personal communication services and visual communication services. Bellcore developed two interesting architectures for distributed processing environments: Open Systems Communication Architecture (OSCA) and Information Networking Architecture (INA) which is discussed below.

The *Open Systems Communication Architecture (OSCA)* architecture enables flexible combination of software products that meet the business needs of PNOs [52]. OSCA was publicly announced in December 1988. The main goal of OSCA is to promote the interoperability of large-scale software products. Software products, as well as their computing environments are often brought into the market independently and have varying degrees of reliability, accessibility and availability. The OSCA architecture is a logical architecture, which is modular and its specification of interconnection and interoperability of software components enables the composition of software systems from different components developed by different suppliers. The OSCA architecture consists of a set of concepts and rules to allow those heterogeneous software products designed within the OSCA architecture to operate as a system of systems in a loosely coupled, distributed configuration. The most important concept is the use of three layers to organise the software in a distributed system (see Section 3.5.1 for details).

The *Information Networking Architecture (INA)* has been developed by Bellcore in '92-'93. The INA concepts have been used by several American PNOs [36]. The INA initiative is driven by the need for PNOs to offer advanced services at a reasonable cost. Systematic design of telecommunications services and management of the software becomes more and more important for telecommunication services, as well as, interoperability between applications deployed in a heterogeneous software and hardware environment. For the introduction of new services in the telecommunications network the need for a modular software structure was identified by INA. INA promotes reuse of existing software components where possible and many OSCA principles have been reused for the INA architecture. Many ideas from INA have been incorporated into the TINA architecture.

The goal of INA is to define a set of concepts and principles for the design and implementation of telecommunications applications regardless of whether the application is deployed on a single computing node or distributed among several computing nodes. INA does not specify any specific telecommunications service but its goal is to define a framework within which such software applications can be specified and designed. INA defines a software architecture that integrates the existing work in TMN, OSI/Network Management Forum, ODP-RM and IN. INA applies an object oriented approach based on the object model of ODP-RM. INA extends the existing efforts where necessary to provide a consistent framework.

### Telecommunications Information Networking Architecture Consortium (TINA-C)

TINA-C is an international initiative that consists of members from the computer industry, telecommunication network operators and telecommunication vendors [6]. The consortium consists of about 40 companies and established a Core Team consisting of about 30 researchers from participating members on a full time basis, located at a single place. The consortium started officially in 1993 and has a duration of five years. The goal of the TINA consortium is to define and validate a software architecture that will enable efficient introduction and management of new and sophisticated telecommunications services. *Auxiliary projects* are established to validate the TINA architecture by means of experiments in labs and field trials. *Demonstrators* have been built to demonstrate and promote the TINA architecture. The first worldwide demonstration was given at the Telecom '95 conference in Geneva in 1995.
The TINA architecture is based on object-oriented technology and distributed computing and incorporates results from international standards such as ITU-T/ISO ODP-RM, ITU-T TMN and OMG's CORBA architecture. The TINA architecture is heavily based on ODP-RM but refines it towards the telecommunication area. The TINA architecture consists of four sub-architectures, i.e. the Service architecture, Resource architecture, DPE architecture and the Management architecture.

- The *Service architecture* [63] defines a set of concepts, principles and guidelines for constructing, deploying, operating and withdrawing of telecommunications information services. The service architecture defines the objects that constitute a telecommunication service, and how these objects should be used and interact;

- The *Resource architecture* [19] defines a set of generic concepts, which describe transport networks in a technology independent manner and provides the mechanisms for the establishment, modification and release of network connections;
- The *DPE architecture* defines the information, computational and engineering concepts [56][60][67] that are used in the other sub-architectures to specify telecommunication services. Additionally, the DPE architecture defines a generic DPE [53], which is used as the platform on which TINA compliant services are built;
- The *Management architecture* [70] provides generic management principles and concepts for the management of services, network and computing platform. As this architecture is outside the scope of this chapter it will not be further addressed.

All these sub-architectures define concepts, rules and guidelines that should complement each other in the trajectory of specifying a TINA service. The concepts defined in TINA are based on results of standardisation bodies. To re-use the work done on DPEs TINA has adopted the work done by OMG on CORBA. TINA has based itself on the work done by ITU-T and ATM Forum on ATM switching and management to support the communication needs for multimedia services.

Section 4.3 focuses on the modelling of multimedia services according to the TINA concepts.

### Microsoft

Microsoft is founded in 1975 and is the worldwide leader in software for personal computers. Microsoft offers a wide range of products and services for business and personal use. Microsoft defines its own solution for the distributed environment that competes with the solution provided by OMG CORBA. The Object Linking and Embedding (OLE) technology forms the basis of Microsoft's strategy to evolve the windows family into object-based operating systems.

The Component Object Model (COM) is the Microsoft answer to the ORB defined by OMG [2]. The COM is the 'object communication bus', which provides a standard communications layer for higher level services, such as the OLE services. COM provides the necessary mechanisms for interoperation between software components. It enables interoperability between components and applications that are written in different programming languages or by different companies and run on different hardware platforms. COM is object-based and adopts the concept of interfaces, which acts as a contract between different COM components. An interface in the context of COM provides a set of related operations, which models the behaviour of an object. Distributed Component Object Model (DCOM) is an extension of COM. DCOM enables components to communicate directly with each other across a network. DCOM defines a standardised architecture-independent wire format and protocol for interaction

between objects on heterogeneous platforms. It is based on the OSF-DCE RPC specification. DCOM provides location transparency between OLE components.

MiTV is based on the OLE/DCOM technology. An overview of the MiTV architecture is given in Section 4.5.

## 3.3 Comparison criteria

This section describes the criteria on which the different consortia described above will be compared. The criteria have been limited to 'object orientation', 'separation principles' and 'multimedia' in correspondence to the prime focus of this thesis, as outlined in Chapter 1.

### Object Orientation

Building distributed applications is a complex task and can be made easier when a modelling process is applied that is able to manage the complexity, by abstracting the irrelevant details or by splitting the distributed application into separate components, which can be dealt with separately. Object orientation provides properties such as encapsulation, abstraction, autonomy, modularity and reuse of components. These properties are important for distributed telecommunication applications since rapid deployment of new or improved services becomes critical for PNOs in a competitive market [28].

Object orientation is also used as the basis for many products that dominate the distributed processing environments such as OMG's CORBA and Microsoft's Windows NT. Applying OO techniques should enable faster service creation by reusing company specific components and applying general 'off the shelf' components as much as possible. Reuse of proven components for service creation reduces errors and increases the overall quality of service offered to the user.

The principles of object orientation claimed to be proven more suitable than other approaches for the design, development and maintenance of distributed systems and applications [28]. This means that an important requirement for consortia should be the support of object orientation in their architectures. The architectures are studied on their (different) OO modelling approach and their support for the structuring of a system for analysis and design. The result of this study is of particular interest for the specification of the case studies in Chapter 6.

### Separation principles

As noted above, distributed systems are complex systems and in order to deal with this complexity several separation principles are applied in the design process to decompose the problem domain into small, manageable pieces. The use of separation principles is applied in many architectures. For example, the separation between analysis, design and implementation is a well-known separation principle. Separation principles can be

coarse-grained such as the ODP viewpoints or rather fine-grained (e.g. separation rules to determine the service oriented interfaces and other non-service interfaces for a computational object).

Applying separation principles will assist the designer in managing the complexity encountered in the trajectory from service requirements to deployment of the service in the existing distributed environment. The most important separation principles used in the architectures developed by the consortia that fit together with the object oriented approach are analysed. This study is useful for the case studies in Chapter 6 that applies these separation principles.

### Multimedia

Most telecommunications services that are being developed for the home and business market are multimedia services: Internet services, video-on-demand, multimedia mail, desktop videoconferencing etc. From studies [59] it is identified that services with a significant multimedia component have an increasing market share and provide an improvement of the telecommunications services offered by PNOs. Multimedia services also increase the traffic over public networks, which is an interesting issue for PNOs. This means that de-facto and de-jure standards should have certain means to deal with multimedia or are able to extend their architecture with multimedia concepts.

## 3.4 Object Orientation

This section compares the object models of the consortia described in Section 3.2. The ODP object model is used as a basis and the main differences of the models of CORBA, TINA, INA and Microsoft COM/OLE are discussed. All models (except COM) are very close to ODP but differences exist. The most important ones are discussed below.

### Interfaces

The current OMG object model supports only one interface for each object. This means that the terms object and interface are synonyms in OMG. In ODP and TINA [60] objects can have multiple interfaces per object, which is valuable for structuring the object's behaviour into multiple system's concern such as 'usage', 'management' etc. The ODP/TINA object model allows (multiple) interfaces to be created and deleted dynamically. It allows an object to support several distinct service operations on a common state. The OMG model does not support this feature and therefore, if an interface is deleted the associated object is deleted as well. It should be noted that existing mechanisms can be used in CORBA to emulate multiple interface objects. In this case an object is regarded as a composite object, which does not actually contain the objects but has references to the objects it contains. However, one common state for the object is not maintained.

The Object Linking and Embedding (OLE) component is the Microsoft object in the ODP sense. An OLE component can support one or more interfaces. An interface is a

strongly typed group of semantically related functions and can be compared to ODP operation interfaces. An OLE component can support one or more interfaces that define a set of related functions. An interface defines an agreement between the OLE components and describes how clients should use it (i.e. the expected behaviour but not the implementation). Every OLE component must support the so-called 'unknown' interface, which allows clients to discover the type of interface supported by the OLE Components. The 'unknown' interface also allows navigation between multiple interfaces of an OLE component, and performance of operations on the OLE component as a whole (e.g. delete component).

A difference with the ODP object model is that the interfaces of an OLE component do not have any state and cannot be instantiated to create a unique object. An interface is simply a group of related functions. The clients are given a pointer to access the functions in that interface, the pointer is not related to state information.

### Stream interfaces

The ODP/TINA object model supports stream interfaces for the exchange of time-based media. Currently, no equivalence is available in OMG that only encompasses operation interfaces.

### Binding

The concept of explicit binding (see Section 2.3.3.3) as defined by ODP-RM and used in TINA is not supported in the OMG model. Using this concept, the application designer can model complex forms of interaction between the objects. The TINA object model details explicit binding for stream interfaces and relates it to their Communication Session Manager (CSM) object that is responsible for the set up, maintenance and release of connections between objects (see also Section 4.2). The use of the CSM object for explicit binding in TINA is tailored towards a telecommunications environment and is not always applicable in other non-telecommunications specific environments.

For the interaction between OLE components, Microsoft defines the Component Object Model (COM), which is responsible for the binding of OLE components. COM connects a client to a server OLE component and allows a client to invoke operations directly on the server. COM can be implemented in several ways. A client application having a pointer to a server interface can call any of the operations defined by that interface directly. If the operations of the server interface are in a local process, COM is implemented using a local procedure call. When the server object and its interfaces are located in a remote process, COM becomes Distributed COM (DCOM) and is implemented using RPC invocations across the network.

### Polymorphism

OMA supports the polymorphism property and therefore operations invoked on an object can cause a different effect depending on the object that receives it. ODP also supports polymorphism. The Microsoft object model does not support polymorphism.

### Interface Inheritance

Inheritance is specified in OMA using OMG-IDL although it is not allowed to override or specialise operations in the subclasses. In TINA, inheritance is related to deriving a new template by modifying an existing template. The Microsoft object model does not support multiple inheritance but instead OLE components can have multiple interfaces and reuse existing interfaces offered by other components through the use of aggregation. This means that an OLE component encapsulates the services of other components and when a client invokes an operation the encapsulated component can in turn invoke an operation on the contained component.

### Templates

TINA uses the templates of ODP as a basis but formalises and extends them using the TINA Object Definition Language. Extensions can be found for the group object template (see also Section 3.5.1). OMG defined OMG-IDL to define the structure of an interface. OMG-IDL is less expressive than TINA-ODL (see Chapter 5). Microsoft defines their own Object Definition Language to describe the interfaces of OLE components. The Microsoft Interface Definition Language (MIDL) is not programming language neutral like OMG-IDL. MIDL is the Microsoft implementation of DCE RPC IDL, which looks like a heavily annotated C header file and assumes the programmer will develop in C. The MIDL compiler generates stub and proxy code and C or C++ header files that define an interface and its functions.

### Building Block

INA slightly refines the object model of ODP and applications in INA are deployed in units of software called building blocks. A building block is a software package composed of several objects and is installed and modified independently of other building blocks. Interfaces offered by objects in a building block, which are externally visible are called contracts. The INA model is primarily concerned with building blocks and contracts and puts no emphasis on the internal structure of a building block. In INA, a building block is deployed on a single machine and cannot be distributed. This is different from ODP were a composed object can be distributed.

### Object model

Many of the consortia described in Section 3.2 use the object model of ODP-RM as a basis. This is the case for TINA and partially the case for INA. In addition, these consortia add certain refinements to the ODP model; TINA is the most advanced one as discussed below.

The OMG object model has been completely adopted by IMA. IMA adds multimedia oriented descriptions to the OMG specifications. IMA provides detailed IDL specifications of interfaces related to the control and management of multimedia devices (see also Section 4.2).

DAVIC has no object model although DAVIC 1.0 documentation states that it uses an object-oriented approach for the development of service elements (see also Section 4.4.1. It is unclear which interfaces are supported by each service element and the encapsulation criteria is not strictly applied meaning that an operation performed in one service element will affect the state in another service element. DAVIC 1.0 specifications are based on a layered non-object oriented model (see Section 4.4). However, the author expects changes in the future releases since DAVIC is planning to use OMG standards for their implementation.

OSF-DCE is based on a procedural model instead of an object-oriented model. OSF-DCE uses the RPC paradigm [48]. DCE is currently including possibilities for distributed object oriented programming such as integrating C++ in its specifications.

The differences between the ODP/TINA model and OMG have been recognised by OMG and through two RFPs 'Multiple Interfaces and Composition' [61] and 'Control of management of A/V Streams' [62]. The features specified by the ODP/TINA object model are put forward to OMG. It is expected that the OMG object model (i.e. CORE 95 model) will be extended with these concepts by early 1997.

## 3.5 Separation principles

Many separation principles exist but only those are discussed that are relevant in the context of this thesis. The TINA life cycle model and OMG OOA&D principles as the 'high' level separation principles are discussed in this section. Section 3.5.1 discusses several separation principles that can be positioned in the computational viewpoint. This viewpoint is of interest for the design of services deployed on DPEs. Finally, Section 3.5.2 proposes a solution that combines several of the separation principles discussed and this solution is applied to model the case studies in Chapter 6.

### TINA life cycle model

TINA introduces a life cycle model that defines an ordered set of processes that can be identified during the development and operation of a (telecommunications) service. The life cycle model is a combination of traditional software engineering methodologies and activities required to operate, use and maintain a service [63]. The TINA life cycle model can be regarded as a framework identifying the different phases related to service development and deployment. Each phase can then be detailed using certain tools and techniques. Figure 3.3 shows the primary phases of the TINA life cycle model. The *enterprise goals* phase describes the overall goals and objectives a stakeholder wants to achieve with the service. The enterprise goals influence all life cycles rather than being

a specific phase in the life cycle. The *development phase* is defined in [63] as "all the off-line activities required in designing and developing the software and any special hardware associated with a service". This phase is the most relevant one for the specification and implementation of services. The development phase is decomposed into several phases needed for the design and validation of a service (see Chapter 6 for more details).

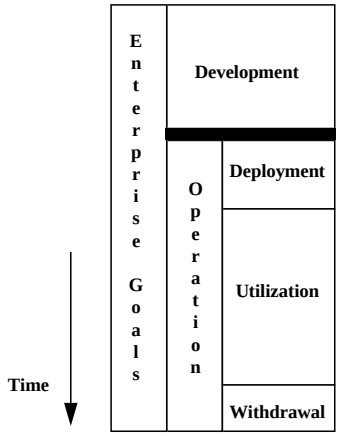



Figure 3.3: TINA Service life cycle model

The operation phase deals with the activities related to deployment, utilisation and eventually withdrawal of a service. Deployment relates to the activities to install the developed software and hardware into an operational environment consisting of computer and network infrastructures. The utilisation phase is related to activities to subscribe users to the service and the actual service usage. Withdrawal is related to all activities needed to deactivate and remove the software and hardware when needed from the infrastructure.

### OO Analysis and Design principles (OMG)

OMG has issued an RFP [152] for the OOA&D of applications, which have to operate in a heterogeneous distributed environment. It is expected that an integration of the object oriented methods of Booch, Rumbaugh (OMT) and Jacobson (OOSE) will become the OOA&D method for CORBA based systems. The combined effort is called the Unified Modelling Language (UML) [151] and is the unification of the Booch, Objectory and OMT methods and incorporating ideas from other methods as well. UML encompasses a number of models that can be used to model and design a (distributed) application:

- *Use case diagrams* are used to organise and model the system's behaviour. In UML it is possible to develop high level use cases (e.g. to communicate with the customers and marketing executives) as well as detailed, very technical use cases to communicate, for example, between engineers;

- *Class diagrams* are used to capture the static semantics of the classes that constitute a system;
- *State-machine diagrams* are used to capture the dynamic semantics of a class;
- *Message-trace diagrams*, *object-message diagrams*, and *process diagrams* capture the dynamic semantics of collaborations of objects;
- *Module diagrams* are used to model the development view of a system;
- *Platform diagrams* model the physical computing topology upon which an application executes.

### 3.5.1 Separation principles in the computational viewpoint

This section discusses the TINA object group, OSCA's three-layer approach and the OMG business object model to organise computational objects. ODP's approach to handle complexity in the computational viewpoint is described by the ODP composition/decomposition principle. The mechanisms described here are quite different and often complementary.

#### ODP Composition/Decomposition

The design process of a distributed application may use *composition* or *decomposition* mechanisms to deal with several levels of abstraction. Decomposition allows a complex distributed application to be decomposed into a number of simpler objects, which can be further decomposed at a lower level of abstraction. The other way around, composition is the process of composing sub-objects into a single higher-level object.

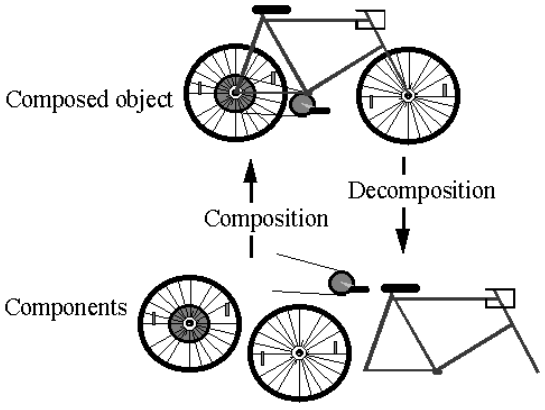



Figure 3.4: Composition/decomposition

Figure 3.4 illustrates the use of four objects, (front wheel, back wheel, frame and cycle mechanism). In order to compose the four components into a single bicycle, the behaviour of the wheels must be defined so that it interacts appropriately with the frame and cycle behaviour. If the wheels, frame and cycle mechanism are composed into one

object (the bicycle), then interactions between the wheels and the cycle mechanism are hidden (they become internal actions of the composite object). The bicycle object provides several external interfaces, which allow the cyclist to operate the bicycle (e.g. steering wheel and pedals).

### TINA's object group

TINA defines the object-group notion to control the complexity of many (computational) objects. It allows organisation of objects into groups that can be managed, installed, maintained and built as a unit. An object group is an encapsulated unit so that only a subset of the interfaces offered by the object in the group are visible outside the group (these are called contracts). One object in the object group is designated as group manager. The group manager plays a central role in an object-group and is responsible for instantiating and deletion of objects, as well as, mediating management operations on the objects part of the object group. The group manager encompasses the policies that are valid for the whole group and has knowledge of the interface types supported by its group objects.

Nested use of object-groups is allowed implying that inside an object group other object-groups might be present. TINA prescribes that the objects part of an object-group must reside in the same capsule (i.e. same computer node and same process) but other (nested) object groups might reside on other computing nodes and different capsules.

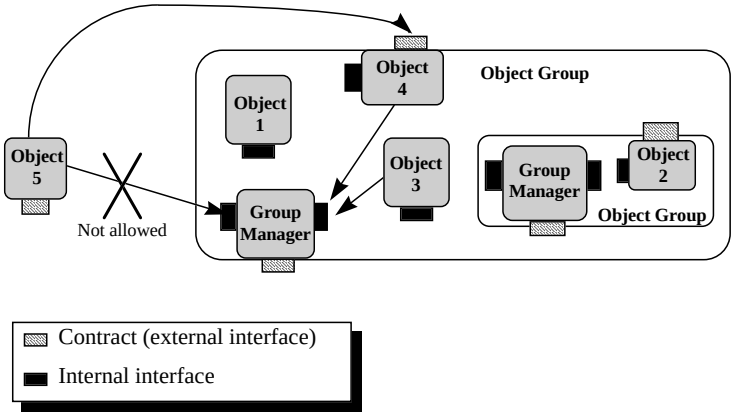



Figure 3.5: Object groups in TINA

The description of TINA object groups is rather similar to that of INA building blocks but it differs in the sense that nested object groups can be distributed over different computing nodes. Another difference is that INA does not apply any nesting of object groups. In INA, a building block is a single software component, which is deployed on one computing node. TINA introduces the notion of group manager, which is responsible for managing an object group whereas INA does not identify such an object. INA is only interested in the contract specifications and not in the internal structure of a building object.

The TINA group object is a refinement of the ODP principle of object composition/decomposition although it adds much more detail to it.

### OSCA three layer approach

The OSCA architecture defined by Bellcore identifies three layers to classify its building blocks (these are identical to the INA building blocks). The functionality offered by a building block must fit in exactly one of the following layers:

- *User-layer* building blocks provide services for end-user (human) interaction with the system's environment. They allow an end-user to access functions provided by the system;
- *Data-layer* building blocks manage the semantic integrity of data (so-called corporate data) that is of importance to the organisation as a whole. This type of building block has access to the corporate data and has the exclusive right to provide access and update functionality;
- *Processing layer* building blocks implement all functionalities, which neither belong to the data-layer nor to the user-layer. This typically includes management and operational functions.

This separation principle proposed by the OSCA architecture should improve interoperability of large-scale software products. Classifying vendor software products into one of the three layers facilitates reuse and replacement of software components. For example, the replacement of a user layer building block (e.g. character text oriented interface) with a more advanced building block (e.g. a windows GUI) is possible without revision of building blocks in other layers.
Interoperability is also achieved since building blocks can only communicate via contracts (enriched interfaces).

The OSCA principles are regarded as a way to classify computational objects into different functional categories. OSCA principles have been applied extensively by KPN Research in several PTT Telecom projects [64].

### OMG Business Object Model

The OMG Business Objects Domain Task Force has submitted an RFP [27] for the specification of so-called *Common Business Objects* (CBO) which represent an entity relevant for a business. A business object models a real world thing, and is implemented by one or more (computational) objects. It is represented as an entity that has a name, attributes, behaviour, relationships with other objects, rules, policies and constraints.

OMG requests for the specification of business objects as high-level application components that are relevant and reusable by application developers. OMG identifies a multi-tiered model to classify business objects. This is rather similar to the OSCA three-layer model and business objects are positioned in one of the following areas (or layers):

- *Presentation layer* objects deal with the presentation and user interface;
- *Business layer* objects represent the (company) business semantics independent of storage or user interface;
- *Persistence layer* objects deal with the storage such as database management systems (DBMS) or storage systems.

The separation of business object into one of these three domains is not mandatory for OMG and alternative proposals (e.g. [149]) have been made to group business objects. Figure 3.6. shows an alternative solution.

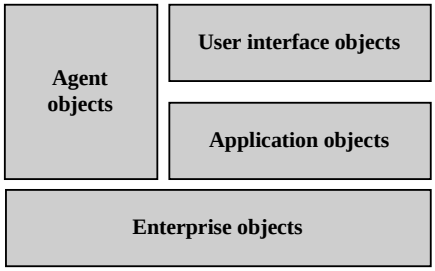



Figure 3.6: Extension of the OMG three tiered model

In this proposal, the *enterprise objects* implement business processes that are shared by many applications in a company. The *application objects* model a particular problem or segment of the business. Many application objects use enterprise objects and there will be a close relationship between the objects of both layers. Agent objects model mechanisms for monitoring and directing activities in the other layers (e.g. workflow management components, expert systems or intelligent tutors). *User interface objects* deal with the presentation and user interface. In this model the persistence objects are not positioned explicitly and might be present in each of the layers.

### 3.5.2 Relation between separation principles

This section shows how the various separation principles discussed above can be related. Figure 3.7 shows this relationship.

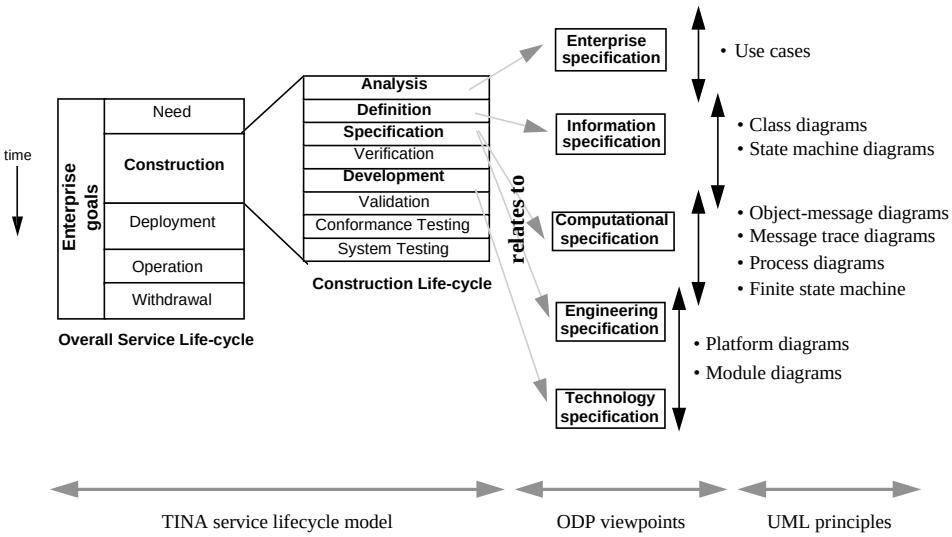



Figure 3.7: Relation between separation principles

The TINA life cycle model is the most general model encompassing the whole trajectory from initial idea till withdrawal of a service from an operational environment. The TINA life cycle model provides general hints of what to do in each phase but details are omitted. This makes the model useful to be an overall road map to identify the various life cycles of a service. The TINA *construction phase* can be detailed assigning the ODP viewpoint languages to the various sub phases of the construction phase as illustrated in Figure 3.7. ODP-RM is not very suitable for the other phases of the TINA life cycle model because it does not define any concepts or rules related to deployment, operation and withdrawal of distributed services operating in an operational environment.

The models defined by UML can be used to refine the viewpoint languages. These models are suitable for the analysis and design of distributed services and thus a refinement of the TINA construction life cycle and ODP viewpoint languages. Figure 3.7 shows how the UML concepts can related to the ODP viewpoint languages. In fact, the various UML diagrams as show in Figure 3.7 are a further refinement (i.e. practical use) of the general concepts as described in each ODP viewpoint language. UML can be actually used by analysts and designers to design applications.

The *use case* model is used to describe the problem domain and has the same purpose as the ODP enterprise language. Based on the use case model, class diagrams and state machine diagrams are derived in UML. These diagrams use the OMT notation technique and are independent of any implementation. These diagrams can be regarded as a practical implementation of the ODP information concepts of static and dynamic and invariant schema.

In UML the *class diagrams* and *state machine diagrams* serve as input for the *object trace diagram*, *message trace diagram*, *process diagram* and *finite state diagram*, which together model the objects and the behaviour between objects. These UML diagrams are a practical implementation of the ODP computational concepts of object template and behaviour between computational objects and the correspondence between the information and computational specifications.

UML is designed to be used in distributed environments and identifies models to deploy objects on a distributed system. Although different terminology is used in UML, the *platform diagram* and *module diagram* are a solution for ODP engineering and technology specifications.

The correspondence described above is a first approximation and it can be argued whether the UML diagrams exactly fit to the ODP viewpoint languages as suggested above. A precise relation is not feasible since ODP and UML have a different basis and have been developed independently from each other.

### Grouping of computational objects

The classification of (computational) objects into domains has been promoted by various groups to obtain optimal reusability, interoperability and independence between objects. In [150] even 6 different layers are proposed to classify computational objects. However, there is no single unambiguous solution to separate objects in layers. Multiple solutions are available as illustrated above, and no standard is agreed upon yet. In KPN also separation principles are used similar to those defined by OSCA. Practical experience in KPN Research has revealed that it is very hard to apply a strict separation of objects into layers. It is, for example, often difficult to classify an object as part of the storage layer or business layer. Also vendor products do not apply the same strict separation principles and combine often objects from different layers into a single larger object. Although it is desirable to apply separation principles to classify computational objects, the author believes that it is unrealistic for the short term to realise a strict separation between objects belonging to different domains since a large installed basis of business objects offered by various vendors is available, which do not take into account those separation principles.

On the other hand, composition and decomposition of computational objects is a broadly accepted mechanism to deal with the complexity of software. Especially in the design phase one starts with coarse-grained objects and during the design process the objects are decomposed in more fine-grained objects.

## 3.6 Multimedia concepts

This section differs in scope from the previous Section 3.4 and Section 3.5 and does not provide detailed information regarding multimedia concepts developed in each consortium. The objective of this section is to justify the choice of the consortia and standards that will then be further detailed in Chapter 4. To select the consortia of interest, their multimedia concepts (if present) were examined and classified along the ODP viewpoints. Distinction is made between consortia that provide generic concepts to specify multimedia, and those that apply specific solutions for multimedia services. The result of this analysis is summarised in Table 3.1.

| Consortia<br>Standards<br>Products | Generic specification of multimedia concepts | | | Specific multimedia implementation |
|---|---|---|---|---|
| | enterprise and information | computational | engineering | |
| **ODP** | ✓ | ✓ | ✓ | × |
| **TINA-C** | ✓ | ✓ | ✓ | × |
| **IMA** | × | ✓ | × | × |
| **MiTV** | × | × | ✓ | ✓ |
| **DAVIC** | × | × | ✓ | × |
| **OMG** | × | × | **No but RFP** [62] | × |
| **OSCA/INA** | × | × | × | × |
| **OSF-DCE** | × | × | × | × |
| **SPIRIT** | × | × | × | × |

Table 3.1: Comparison of multimedia concepts in standards and consortia

Both ODP and TINA have included generic[5] multimedia concepts in different viewpoints (see Table 3.1). This is not surprising since TINA is a specialisation of ODP-RM for telecommunications. TINA has similar concepts as ODP-RM but differences exist that are detailed in Chapter 4. Neither ODP nor TINA has detailed the multimedia concepts by adopting specific multimedia standards that implement these concepts. Particularly for TINA, one might have assumed the adoption of specialised multimedia standards suitable for the telecommunications domain.

The IMA and MiTV architectures provide multimedia concepts that are more targeted to specific DPEs. IMA proposes specific control interfaces for multimedia applications and specifies the properties of time-based flows. MiTV proposes an interesting solution for the modelling of synchronisation of time-based flows. MiTV also uses a specific implementation environment for their multimedia services. Both Microsoft and DAVIC provide different engineering solutions for Interactive Television and they will be

5 Generic means here that the multimedia concepts are not linked to a particular implementation or product and can therefore be used by a wide range of multimedia services in a heterogeneous environment. On the opposite, specific multimedia concepts can only be used in a particular environment (e.g. a proprietary implementation).

detailed in Chapter 4 to obtain insight of how distributed multimedia services are implemented in real systems.

OMG does not yet support multimedia but a RFP for the support of stream interfaces has been initiated. This implies that OMG will support multimedia applications in the near future. Other consortia neither specify specific concepts related to multimedia applications nor support multimedia applications.

## 3.7 Towards hybrid standardisation

The previous sections described several approaches and solutions to design and implement distributed multimedia applications. The result of all these efforts is mutually incompatible architectures and there exists a lack of open interfaces, which prohibits interoperability of multimedia services in an open heterogeneous environment. To solve this problem a general framework for these standards is needed, which can combine the results of de-facto and de-jure standardisation in a consistent manner.

This section outlines the author's view of how this could be realised using ODP-RM as a basis [83]. The ODP-RM is the de-jure standard for open distributed processing. It is a general framework that allows a number of different approaches to realise ODP specifications. This flexibility is necessary if the ODP framework is to have a reasonable lifetime, incorporating new developments as they come. To achieve this flexibility, the ODP workgroup identifies additional categories of standards that will complete the ODP reference model (see Figure 3.8).

The ODP workgroup identifies so-called *architectural frameworks*, which complement ODP-RM in specific areas. Examples are the security framework, QoS framework, and Naming framework. Additionally, *abstract generic components* are defined that are standards for certain functions (e.g. management functions). The ODP workgroup identifies *concrete components*, which correspond to particular realisations of abstract generic components. In addition to these standards several *notations* linked to the viewpoint languages will be adopted (e.g. IDL, FDTs).

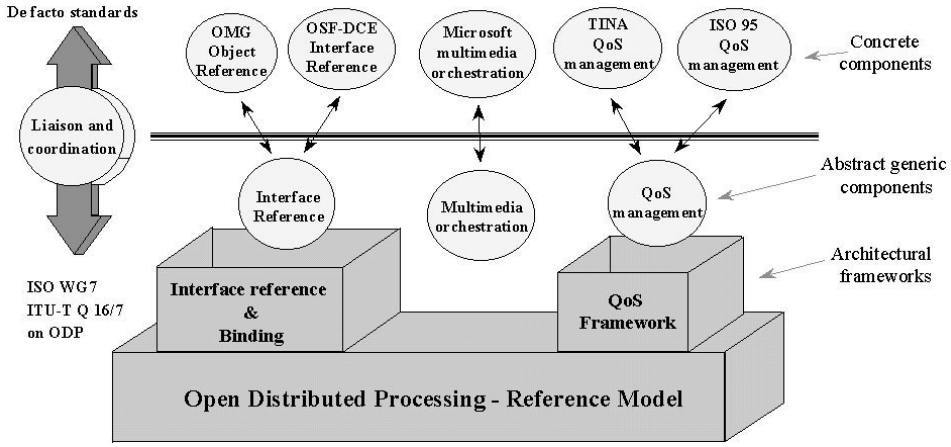


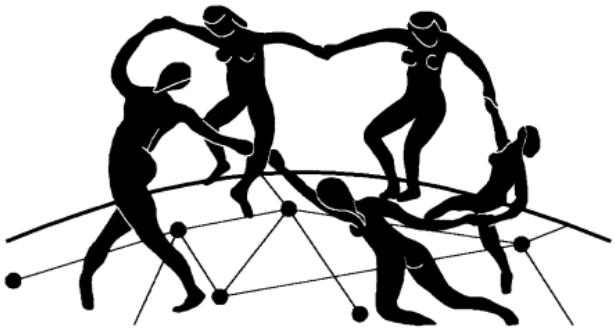

Figure 3.8: Towards hybrid standardisation for open distributed processing

The author believes that the use of ODP-RM as a basis for standards related to distributed processing is an appropriate approach. De-jure standardisation organisations (ISO/IEC and ITU-T) are in a position to build general frameworks, which establish

coherent perspectives for standard development. De-facto standardisation efforts are restricted to the provision of technical components that lack often overall consistency and their lifetime and evolution is limited. However, it is also recognised that de-facto standards are more popular than de-jure standards because of the fast time frame to standardise specifications and accompanying implementations. Promotion of de-facto specifications as de-jure standards, when they are compatible with the ODP reference model, is more efficient than developing the component standards (again) by de-jure standards bodies (e.g. the OMG-IDL language adopted by ODP-RM). Figure 3.8 shows a possible strategy of the ODP-workgroup to incorporate de-facto and de-jure standards.

It is the authors' opinion that hybrid standardisation (de-jure and de-facto standards) of DPEs is the best way forward for ODP-RM. It offers a win-win situation in which de-facto standards provide a long-term framework, and de-jure standards provide up-to-date specifications of components in a reasonable time. Distributed systems based on a long-term (implementation and technology neutral) architecture can evolve and preserve the investments made in distributed systems. In due time, components can be replaced with newer versions without major modifications to the system architecture.

This thesis applies the philosophy of hybrid standardisation and the author wants to incorporate results of consortia into ODP-RM. Where appropriate, it is desirable to influence consortia with ODP-RM concepts. In the area of multimedia no effort is made yet in the ODP workgroup to include results from de-facto standards. In Section 3.6 the appropriate consortia that specify distributed processing environments are identified, which are able to support multimedia services. Chapter 4 details the multimedia specific components. Chapter 5 proposes additions and refinements of the ODP modelling concepts suitable to support multimedia from an ODP perspective that partially include results of de-facto standards.



# 4 Multimedia in Distributed Processing Environments

*Several consortia exist that specify distributed processing environments supporting multimedia services. Four of them (TINA-C, IMA, DAVIC and Microsoft) described in this chapter have the potential to be accepted by a large community both in the telecommunications and the computing industry. This chapter describes and analyses the architectures developed by these consortia focusing on the multimedia concepts. The multimedia concepts found in these architectures are positioned with respect to the ODP information, computational and engineering viewpoints to enable better comparison and future integration. This chapter is a starting point for the inclusion of multimedia concepts in the ODP computational and engineering viewpoint languages.*

## 4.1 Introduction

This chapter gives an overview of concepts and terminology currently available in the IMA, TINA, MiTV and DAVIC specifications. These architectures are selected based on the inventory performed in Chapter 3. The specifications of these consortia contain interesting components worthwhile being compared and related to the multimedia concepts in ODP-RM. As seen in Chapter 2, ODP-RM defines several concepts for multimedia applications but it refrains from details. The focus is on international consortia since the author believes that those groups have a major influence and a wider acceptance degree in the area of multimedia in distributed environments. Also a tendency is that results from EURESCOM or ACTS projects are used to influence these consortia where possible.

The purpose of this chapter is to investigate how multimedia is incorporated and modelled in IMA, TINA, MiTV and DAVIC specifications. In order to compare these specifications and also to incorporate their features in the ODP multimedia framework (Chapter 5) whenever appropriate, the multimedia components of the four specifications are positioned along the information, computational, engineering and technology ODP viewpoints.

Section 4.2 and Section 4.3 investigate the concepts available in the IMA and TINA architectures respectively. These architectures are product-independent and are not biased towards a particular technology. In Section 4.4 and Section 4.5 the concepts of the MiTV and DAVIC specifications are described respectively, which are two different technologies for mass-multimedia applications. MiTV is a specific product of Microsoft whereas DAVIC is product-independent but incorporates existing technologies such as MHEG and MPEG. It should be stressed that MiTV and DAVIC specifications are not stable yet and considerable changes are foreseen. Nevertheless, it is interesting to study how actual implementations deal with multimedia.

## 4.2 Interactive Multimedia Association

IMA developed the Multimedia Systems Services (MSS) specifications. The MSS specifications provide detailed guidelines for manufacturers and applications developers to design applications that are platform neutral. The MSS recommendation consists of three parts where Part 1: ‘functional specification’ [32] and Part 2: ‘Multimedia devices and formats’ [33] are mature. Part 3: ‘Transport and media stream protocol specifications’ [34] is incomplete. This might imply that IMA is not active anymore or relies on the outcome of other consortia with respect to multimedia protocols. The material described in IMA Part 1 and Part 2 is of interest for the multimedia concepts.

### 4.2.1 Concepts related to the computational/engineering viewpoint

The services described by MSS can be regarded as a computational specification of an application since the object oriented approach is applied to specify the multimedia objects in MSS. The interface specifications of MSS objects inherit a subset of OMG’s object capabilities such as the OMG life cycle operations (e.g. create, copy and delete operations).

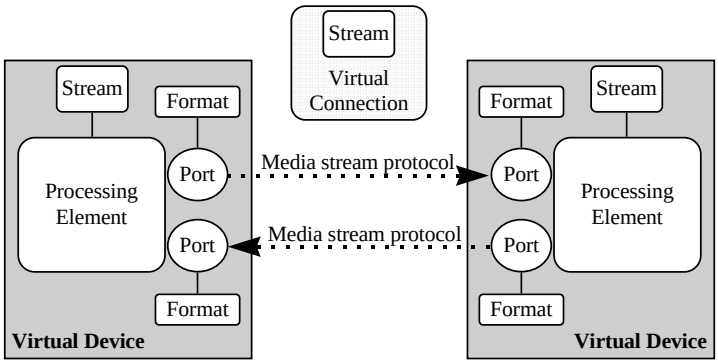



Figure 4.1: IMA concepts for modelling multimedia applications

The MSS architecture defines the following objects (and the objects that derive from it):

- *Virtual device* object abstracts from a wide variety of physical devices, which might be supported by many kinds of operating systems. The *Processing Element* determines the actual functionality of the virtual device (e.g. whether it is an audio or video device). The virtual device supports several generic interfaces for resource management, stream/position control, media/format control and virtual device specific control;
- *Port objects* are part of the virtual device and correspond to the input and output of the virtual device. Ports have a certain format described by the *format object*, which provides an abstraction of the data type information (framesize, encoding etc.);
- *Stream object* provides the client of the virtual device with an interface to observe the position in a time-based flow. Particular stream objects also provide an interface for controlling the time-based flows and other stream objects provide synchronisation interfaces;
- *Virtual connection* represents an object that provides an interface to create a connection between an output port of one virtual device and an input port of another virtual device. The virtual connection object abstracts from the transport connections between the virtual devices to be connected.

Figure 4.2 shows the relationship between the IMA and ODP-RM model to enable comparison. The virtual device relates to an ODP computational object that encapsulates the device specific processing capabilities as defined in IMA's Processing Element. The IMA concept of Port is similar to a flow that is part of a stream interface. A Port has a certain format, which corresponds to the flow characteristics in ODP-RM. The virtual device provides interfaces for the control of time-based flows. It has a format interface to dynamically change the flow characteristics. Compared to ODP this is similar to the control interfaces part of a computational object.

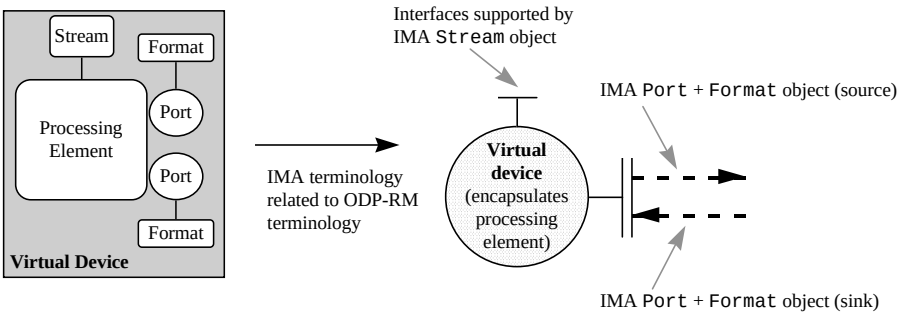



Figure 4.2: Relationship between IMA and ODP concepts

### Virtual connection object

The virtual connection object in IMA is similar to a binding object in ODP-RM (see Section 2.3.3.3). IMA supports point-to-point uni-directional binding (e.g. microphone to a single speaker) and point-to multipoint uni-directional (e.g. microphone to two speakers). This is a limited form of binding as allowed in ODP-RM.

The virtual connection object performs several checks before the source and sink ports of two virtual devices can be bound:

- Type of media that is to be transported between the virtual device ports: both the source and sink ports have a compatible format (expressed in the Format object associated to a port), before data can be exchanged. The virtual connection object can perform negotiation between the source and sink ports to determine common format characteristics. It queries the capabilities of both the source and sink port to find a match;
- The virtual connection determines the appropriate connection between the virtual device ports, which can be for instance a hardware or a network connection. The virtual connection determines if the virtual devices are in the same address space, in separate address spaces but on the same system, or on separate systems. This information is used to determine the optimum type of connection to be made. In case a connection has to be made between separate systems, a so-called virtual connection adapter is used. In ODP-RM terminology this is an interceptor object, which is responsible for format conversion between the systems (if required). A virtual connection adapter can be implemented using various network transport protocols (e.g. TCP(UDP)/IP, OSI TP4, ST-II);
- The virtual connection determines QoS compatibility between the virtual device ports. IMA defines the following QoS parameters that can be negotiated through the virtual connection object: the type of the connection (i.e. guaranteed, best-effort, no-guarantee), reliability whether data loss is allowed, and the maximum and minimum values of the delay, bandwidth and jitter;

- The virtual connection supports synchronisation and data exchange mechanisms. The virtual connection determines if the virtual devices can agree on a common data exchange mechanism. To synchronise two time-based flows, the virtual connection supports a `ControlledStream`-interface containing various operations to control flows.

### Multimedia related interfaces

The MSS provides detailed IDL specifications for the interfaces supported by the objects and uses several OMG Common Object Services (e.g. Life cycle, Event service).

The IMA `Stream`-interface[6] provides inquiry and control operations related to the progression of time-based flows. The operations are independent of specific characteristics of the time-based flow. To obtain the position of a sample (e.g. an audio sample or video frame) in a time based flow, several options exist, depending on the characteristics of the virtual device that supports the `Stream`-interface:

- Virtual devices exist that do not understand the structure of the time-based flow. In this case the virtual device can only report the amount of bytes counted since the time-based flow started;
- The virtual device can interpret the samples (i.e. a certain amount of bytes constitute one sample) but not how the samples translate into flow time. In this case the object reports the stream sample count;
- The virtual device can interpret the data units and is able to derive the flow time. In this case it reports the flow time.

The IMA `Stream`-interface contains operations, which allows the client to observe the flow position. The client can also insert markers in the time-based flows on request for the purpose of monitoring the flow. The client can also ask to be notified when a certain position in a flow has been reached.

The `ControlledStream`-interface is used by the virtual connection object to control the advance of a time-based flow. Five operations are defined:

- The `pause` operation to suspend the flow. This operation is invoked on both the sink port and source port;
- The `resume` operation allows the flow to advance;
- The `mute` operation mutes the stream at the position specified. The flow continues to run (i.e. flow time advances) while the stream is muted;
- The `drain` operation drains the internal buffers of the virtual connection object. Data goes out of the virtual connection object but does not flow into it;

---

6 The IMA stream interface is an operation interface in ODP terminology and not an ODP stream interface.

- The `prime` operation can be thought of as pre-roll operation. The virtual device object fills its internal buffers in anticipation of a `resume` operation. Data flows into the virtual device object but does not flow out of it.

The `SyncStream`-interface contains operations to support the synchronisation between time-based flows inside a single virtual device or between multiple virtual devices. Synchronisation in IMA is based on the use of the master/slave principle where one time-based flow is appointed to be the master and others follow the master. Different ways to achieve synchronisation are possible in IMA. For example, the slave flow may poll the master flow regularly to obtain its position with respect to the master flow. Another option is that the master flow regularly sends events (e.g. time stamps) to the slaves. From these events a slave flow can determine its position and perform adjustments if necessary.

Synchronisation events can be given by the server who informs the client when the alignment between the master and slave flows crosses certain boundaries (e.g. to maintain lip-synchronisation, audio and video flows should be within a tight range). Synchronisation events allow the client to detect synchronisation exceptions and to take appropriate actions.

### 4.2.2 Discussion on IMA

The ODP concept of a binding object is stronger than the virtual connection in the MSS specification. MSS specifications are limited to simple types of bindings whereas the bindings of ODP can be arbitrarily complex. Nevertheless, the IMA specification of a virtual connection can be seen as a concrete means to implement an ODP binding object.

The ODP notion of stream interface generalises the notion of port in the MSS specification. The MSS specification does not fulfil the requirements for stream interfaces that could be manipulated in the same way as operation interfaces. Ports in the MSS specification are not viewed as interfaces and the MSS specification does not explain how ports can be implemented and accessed.

Proper computational models are not present in MSS, which makes the specifications hard to read. Also the distinction between interface and object is not always clear. This is due to the fact that MSS adopts the OMG specification as a basis where an object is equivalent to an interface (see discussion in Section 3.4). This is not always appropriate for the MSS specifications since objects (e.g. virtual device) support multiple interfaces. It would be favourable for future releases of the MSS specifications to align themselves to the ODP computational model and in addition develop engineering models as well, which are currently not present in IMA specifications.

IMA specifications provide a good source for IDL specifications related to multimedia. The question is, however, to what extent these (rather detailed) specifications are

acceptable by other consortia. For example, MiTV uses its own synchronisation mechanism for time-based flows and does not use specific IMA synchronisation operations. When OMG adopts the concept of stream interface, and more generally, the support for multimedia applications, it might be a good opportunity of IMA to bring forward the MSS specifications.

## 4.3 Telecommunications Information Networking Architecture

Specific multimedia concepts in TINA are the stream interface, binding object and stream interface reference. TINA concepts are very close to those defined in ODP-RM though details have been added and separation between viewpoints is less clear. These refinements to ODP-RM are discussed in Chapter 5. In addition to these concepts, TINA defines a model, which is used for the subscription, access and usage of a service by one or more users. Subscription and access are modelled by the Access Session Model [63], which is not further discussed in this thesis. A Service Session Model models the usage of a service. This section focuses on the service session model since it models connectivity between services [11].

TINA defines a general *session model* for all services that operate in a TINA compliant environment. A session model specifies the associations between the members that constitute a service (e.g. video-on-demand service). In the computational viewpoint this means that a session describes the relations between computational objects that are involved in a particular service. TINA distinguishes between so-called service session models and communication session models (see Figure 4.3).

The *service session* model describes the control and connectivity relations between the members of a service, e.g., end-users, network operators and service providers. The *control relations* describe which member can or cannot control the service (e.g. in a conferencing service, which user can or cannot invite other users). *Connectivity relations* describe the possible bindings between stream interfaces for a particular service (e.g. a multipoint-to-multipoint connectivity for a conferencing service). The control and connectivity relations of a service are described by a *Service Session Graph (SSG)*, which abstracts from specific connectivities[7] that are used to realise these relations. Information described by the SSG could be, for example the owner of a session; who is allowed to manipulate the session; and which type of connectivity is supported by the service. The information described by a SSG is represented as an information attribute (i.e. a parameter) in operations used by computational objects to control a session.

7 For instance, the SSG may describe a multipoint-to-multipoint connectivity relation, which is implemented using a set of point-to-multipoint connectivities offered by a network.

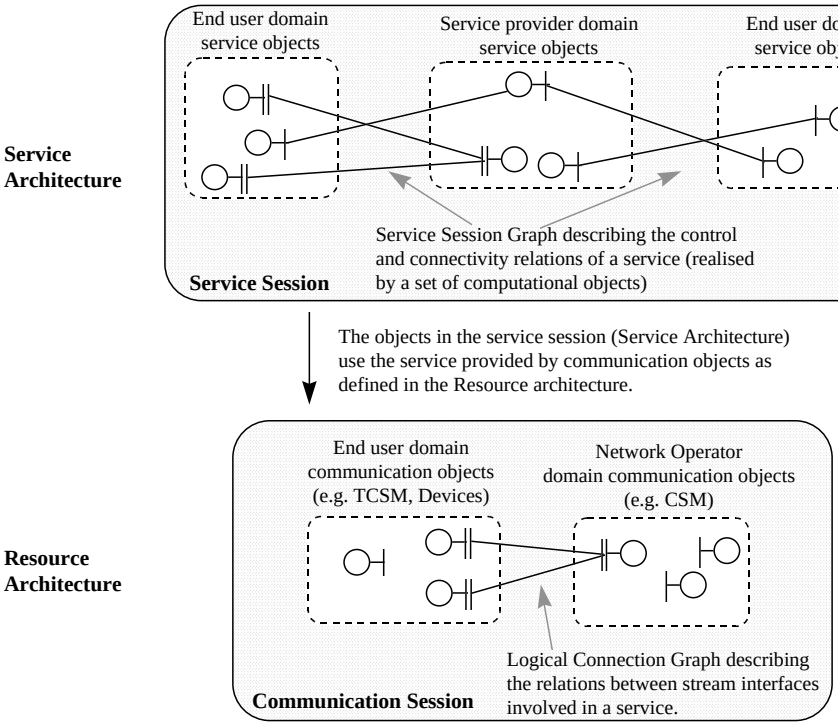



Figure 4.3: The service session and communication session concept in TINA

A *communication session* is used in TINA to give an abstract view on the communication resources and connectivity relations needed by the computational objects part of a service session. Connectivity relations of a communication session are described using a *Logical Connection Graph (LCG).* The LCG describes the connectivity relations between stream interfaces. A difference between the connectivity described by the LCG and SSG is that the SSG supports more complex connectivity relations than those described by the LCG. For instance, the LCG does not support multipoint-to-multipoint bi-directional relations.

The concepts introduced above have been developed in several TINA sub-architectures but are interrelated (see also Section 3.2). The service session model and SSG have been defined in the TINA Service architecture. The communication session and LCG are defined in the TINA Resource architecture. This means that the service session model and SSG concept use the communication session and LCG concept to realise connectivity. Thus, objects in the Service architecture only use the SSG to manipulate the connectivity and control relations. However, to actually set up a stream binding between objects in the Service architecture the LCG is used to describe the connectivity relations between objects in the Resource architecture.

### 4.3.1 Concepts related to the information/computational viewpoint

**Service Session Graph**

Figure 4.4 shows a simplified service session graph (SSG) model focusing on the multimedia features (connectivity control relations are omitted). The SSG in Figure 4.4 represents a general model of possible connectivity relations between stream interfaces and stream bindings that can be established in a service session.

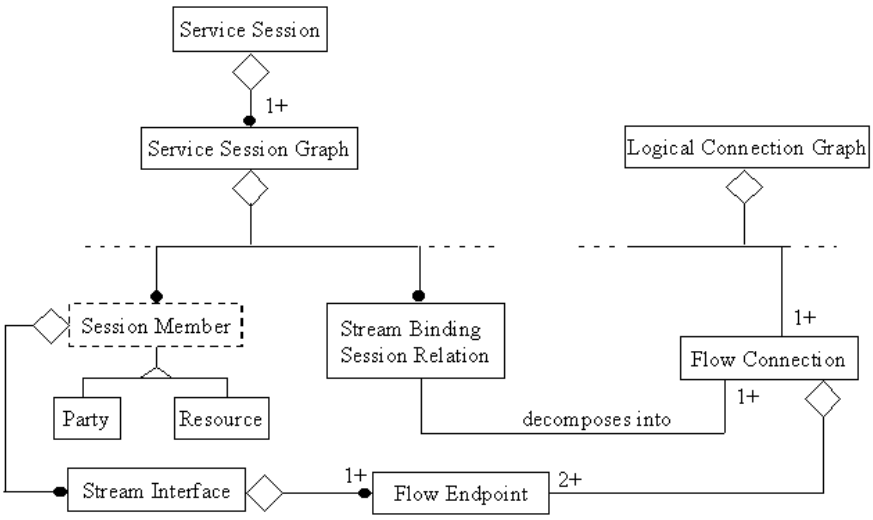



Figure 4.4: Relation between SSG and LCG to express connectivity

The *Service Session* class contains all the information necessary to model the configuration of a session and the relationships between the elements of the session (i.e. users, resources) at a certain point of time expressed in one or more *Service Session Graphs*.

The *Session Member* class is an abstract class (depicted as dotted lines and can not be instantiated), which is further refined into a party class and resource class. The *Party* class represents the users involved in the session that can be either the end-user or service provider. The *Resource* class represents the hardware and software resources involved (or needed) in the session. Resources are for instance, a video bridge, a file, a shared pointer and so on.

The session member can have zero or more stream interfaces for the exchange of time-based flows. A *stream interface* class consists of one or more *Flow Endpoints* that can be either the source or sink of a flow.

The connectivity relations describe how the time-based flows of the members are related. These relationships are expressed in the *stream binding session relation* class.

This class contains attributes that describe the possible relations between the source and sink flows that are part of a stream interface. TINA defines five different topologies, which conform to the ITU-SG11 specifications for the exchange of audio-visual data. These topologies can be either:

- Bi-directional point-to-point (e.g. telephone conversation);
- Uni-directional point-to-multipoint (e.g. near video-on-demand) ;
- Uni-directional multipoint-to-point (e.g. video bridge receiving all audio-visual flows);
- Bi-directional multipoint-to-multipoint (e.g. 3-party videoconferencing where all can send and receive what is sent);
- Bi-directional point-to-multipoint (e.g. interactive video on demand where only the source can receive data)

TINA defines the stream binding session relation class to model complex topologies for time-based flows that were not possible to express in the LCG concept, which was introduced earlier in time by the Resource architecture [19]. Since TINA sub-architectures should complement each other, a relation exists between the SSG and LCG: the stream binding relation in the SSG is supported by one or more LCGs (see Figure 4.4).

### Logical Connection graph

The LCG concept is introduced in the Resource architecture to express the connectivity between end-points independent of how this is achieved, and independent of the underlying network technology. The LCG is able to express point-to-point bi-directional and point-to-multipoint uni-directional connectivity relations, which is more limited than the SSG, which supports all five connectivity topologies described above.

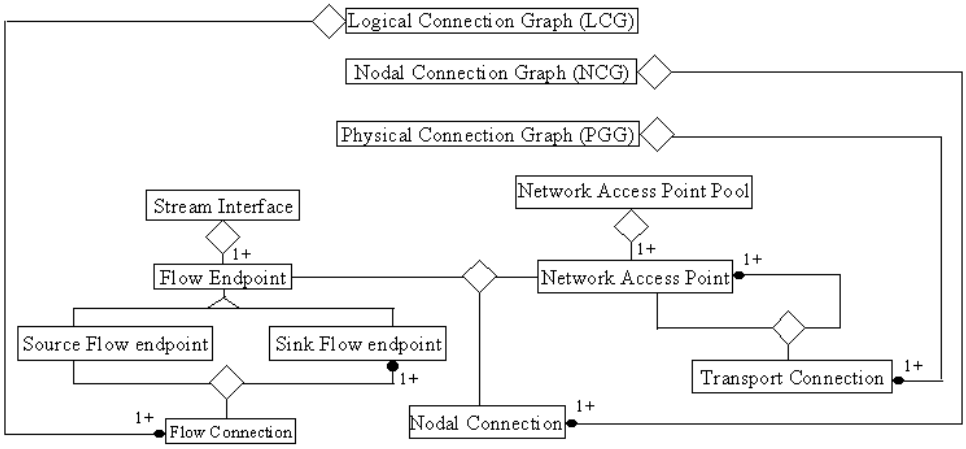



Figure 4.5: OMT model to express connectivity in TINA

A LCG is an information object that represents a single stream binding. The LCG contains one or more flow connections. A *Flow Connection* is the representation of a binding between a source flow endpoint and a number of sink flow endpoints.

### Nodal Connection Graph

A *Nodal Connection Graph (NCG)* is an information object that represents a binding inside the end-user node and it contains one or more nodal connections. A nodal connection is the representation of the connection between a flow endpoint and network access point (Figure 4.5 and Figure 4.6).

### Physical Connection Graph

A *Physical Connection Graph (PCG)* is an information object that represents a transport network binding and it contains one or more transport connections. A *transport connection* is the representation of a bi-directional connection between one or more network access points (Figure 4.5 and Figure 4.6).

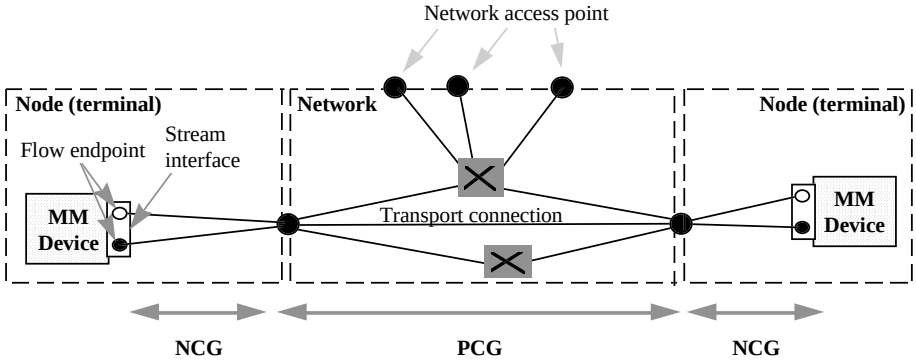



Figure 4.6: Example of relation between TINA connection graphs

These different connection graphs are used to express connectivity relations for stream interfaces at different abstraction levels. Different objects as illustrated in Figure 4.7 use the connection graphs.

## 4.3.2 Concepts related to the computational/engineering viewpoint

TINA defines several generic objects that should be present in each TINA service. One of them is the *Service Session Manager (SSM)* object. The SSM object is the central point of a service session and it co-ordinates and serves all requests from members, which participate in the service session. The SSM object stores the SSG and has the knowledge about connectivity relations between members. It is able to manipulate the SSG through interfaces provided by the SSM object (e.g. removing a member from an ongoing session). The SSM object is also capable of decomposing the SSG into one or more LCGs that are sent to the *Communication Session Manager (CSM*) object. Manipulation of

connectivity relations is achieved by sending updated LCGs to the CSM object, which contains the LCG of a particular session. Additional computational[8] objects are shown in Figure 4.7 and explained below.

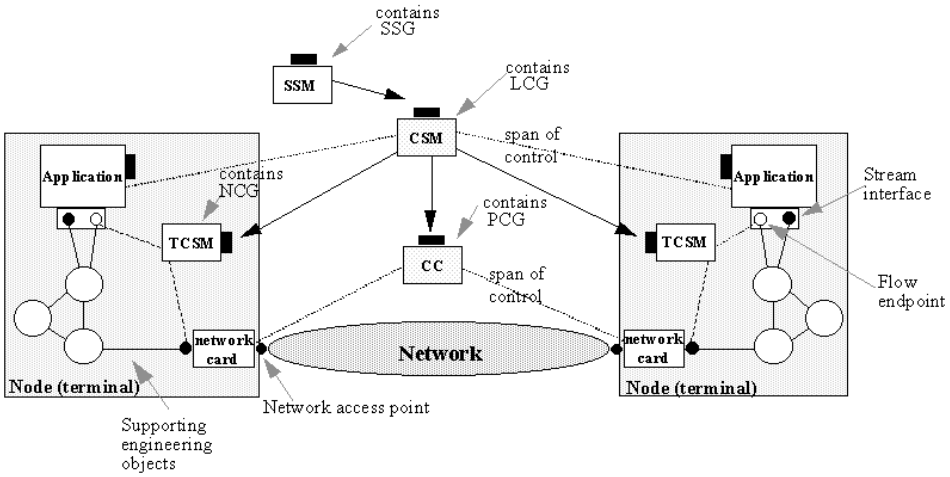



Figure 4.7: Computational/engineering view for modelling connectivity

The CSM object is responsible for the end-to-end application communication. It provides the binding between flow endpoints (depicted by the dotted lines in Figure 4.7). The LCG is used by the SSM object to describe and express to the CSM object the characteristics and configuration of the binding between the flow endpoints in the context of a communication session. The CSM will then decompose the requested binding (i.e. LCG) into two graphs: the NCG and PCG.

The CSM object requests the *Terminal Communication Session Manager (TCSM)* to take care of the NCG. The TCSM object is responsible for the binding inside a user's node. It binds the flow endpoint (typically a multimedia device) to the network access point (typically a network card). The characteristics and configuration of the nodal binding is described by the CSM by means of the NCG within the context of a terminal communication session.

The characteristics and configuration of the transport connection is described by the CSM by means of the PCG within the context of a transport connection session. The *Connection Coordinator (CC)* is responsible for the end-to-end transport connection provided by a transport network, i.e. from network access point to network access point.

8 TINA does not make a clear distinction between the computational and engineering viewpoints and applies a mixture of these viewpoints. The computational objects involved in the set up of streams are positioned in the engineering viewpoint since location transparency is not assumed for these objects. Especially the objects located in the end-user node are located physically in one system.

### 4.3.3 Discussion on TINA

The previous section discussed important concepts currently available in the TINA architecture (as of September 1996), which differ from the ODP concepts to model multimedia characteristics of distributed services. Concepts are mainly situated in the computational and engineering viewpoint even though the OMT graphical notation is used. The various graphs in TINA are used for different abstraction purposes, for instance:

- the session graph (SSG) is an end-user perspective on connectivity relations between time-based flows, abstracting from technical details to set up end-to-end connection;
- the logical connection graph (LCG) is a simplified session graph only describing the end-to-end connectivity relations between stream interfaces;
- the physical connection graph (PCG) describes the transport connections (inclusive link-to-link connections) between network access points in a telecommunications network to create end-to-end connections;
- the nodal graph (NCG) describes a rather engineering/technology view of a connectivity relation within a node.

The different connectivity views are presented in a single information model to show their relationship. TINA adopts a hierarchical approach to connection set up (starting with SSG up to PCGs and NCGs). This is suitable for a service model where all interactions occur via a service provider (SSM-object). In a point-to-point communication scenario between end-users that do not need a third party service provider, the TINA session model is inappropriate. An important starting point in the session model is the fact that a third party (service provider) is responsible for the connection set up (SSM and CSM objects) and not the end-users themselves. This might cause problems since connection establishment is initiated by the end-user without the explicit involvement of a third party. Prototypes [65] implementing (part of) the TINA architecture show that connection establishment is made using signalling protocols (e.g. Q.2931 for B-ISDN) where the end-user initiates connection set up. This implies that the TINA session model is appropriate for services where a third-party (service provider) is needed that requests more complex connectivity but the session model is not optimal for connection set up without the use of a third party service provider (e.g. in LAN environments).

Another issue is the use of nodal graphs to set up a binding between the network access point and (engineering) objects in the end-user node. This is not the correct way since it assumes that a third party (the CSM object is owned by a third party provider) has knowledge about the objects residing in the end-user node. It is the author's view that this is undesirable and it is believed that node internals should not be visible to the outside world. To overcome this problem an alternative solution is proposed in Section 5.2.3 to set up a binding between objects, which does not assume knowledge about the internals of an end-user node by third parties.

A major problem with TINA '95 specifications is the inconsistency between the modelling concepts and terminology in the TINA sub-architectures. Each TINA sub-architecture has introduced its own concepts to model connectivity for time-based

flows. This resulted in the undesirable situation that solutions provided by the Service architecture did not fit to the solutions of the Resource architecture, and the concepts provided by the DPE architecture were even bypassed. To solve this problem the author proposed in [66] suggestions for mending these inconsistencies and advocated the use the ODP concepts for modelling multimedia services. This has already resulted in adaptations of several concepts in the Service and Resource architecture. It is expected that by the end of 1997, consistent terminology and concepts will be used by TINA, based on material presented in Chapter 5.

## 4.4 Digital Audiovisual Council

Applications addressed by DAVIC have an important audio-visual component and are targeted for a mass market. Examples of DAVIC applications are: Movies on Demand, Teleshopping, Broadcast, News on Demand, Telework and Games.

Figure 4.8 shows a simplification of the DAVIC System Reference Model (DSRM) and depicts the type of systems that are described by the DAVIC 1.0 specification [38]. DSRM consists of four basic systems: the *Content Provider System (CPS)*, the *Delivery System (DS)*, the *Service Provider System (SPS)*, and the *Service Consumer System (SCS)*. The Delivery System interconnects the other systems.

DSRM shows certain similarity to the OSI reference model and re-uses the OSI layering approach. A DAVIC system (e.g. service consumer system) is composed of several *service layers*, each providing a certain functionality. Several OSI-RM layers are combined in a DAVIC system in a single service layer (e.g. session and transport layer). On the other hand, the OSI-RM application layer (layer 7) is split in a DAVIC system into a principle service layer and application service layer (see Figure 4.8).

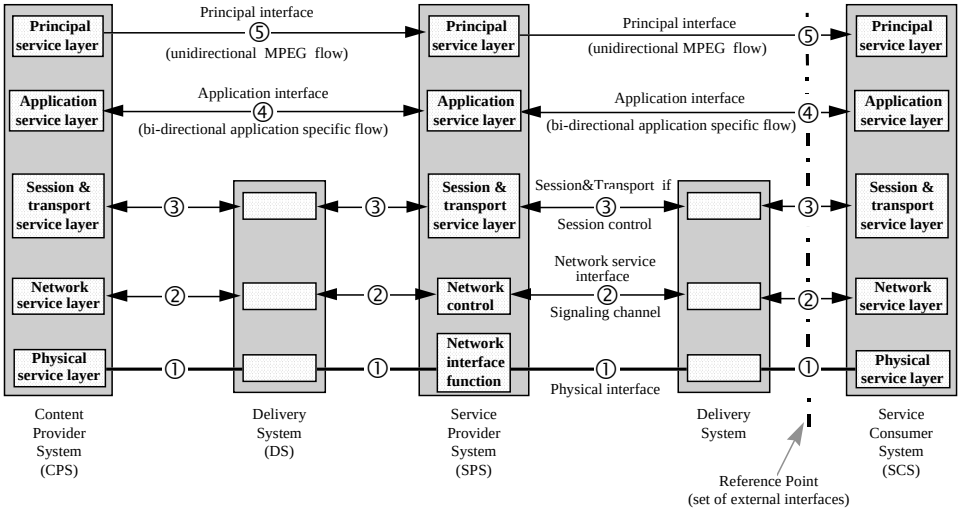



Figure 4.8: DAVIC System Reference Model

A service layer contains one or more objects, and a service layer provides external and internal interfaces to other service layers.

- Internal interfaces are used for the communication between service layers within a single system through APIs. A service layer in a DAVIC system can request services from an adjacent lower service layer through an API. This is similar to the Service Access Point (SAP) concept defined in OSI-RM;
- External interfaces are used for the exchange of information between objects located in the same service layer but located in different systems. Standardisation of external interfaces achieves interworking between DAVIC compliant systems.

*Reference points* are identified between systems to which a DAVIC compliant specification should conform. A reference point in DAVIC represents a set of (external) interfaces through which peer objects (i.e. objects located in different systems but the same service layer) transfer information. The type of information passed through the interface depends on the service layer in which the objects that communicate are positioned. The following interfaces are defined between two systems:

The *Physical interface* (Figure 4.8, ①) where the physical characteristics of signals are defined that are used to represent information. The physical characteristics of the channels, which carry the signals, are also defined. A physical interface is an external interface, which is fully characterised by its physical and electrical characteristics. The physical interface transports all the information offered by the other service layers in a single system.

The *Network interface* (②) is a logical interface between two peer objects located in the network service layer in different systems. The network service layer objects to exchange control information use it. The network interface is used, for instance to

exchange signalling information between the system (e.g. service provider system) and the delivery system. Examples include messages to establish or release connections and to communicate addresses, port information and other routing information.

The *Session and Transport interface* (③) is a logical interface between two peer objects located in the session and transport layer. It is used to exchange control information to set up, and manage sessions that are established between the service provider system and service consumer system. A session defines a higher level association of resources across multiple networks. The interface is used to establish, modify, and terminate a session or to negotiate resource requirements.

The *Application interface* (④) is a logical interface between two objects located in the application service layer. It is used to exchange control information but in this case the information will not be interpreted by the delivery system. The operations of this interface are typically service dependent and are, for instance: to invoke VCR type of commands; the service provider providing billing information.

The *Principal interface* (⑤) is a logical interface between two objects located in the principal service layer. It is used to exchange content information (e.g. audiovisual data) from a source object to a destination object. The content information will not be interpreted by the delivery system. The content information flow is a uni-directional flow from service provider to service consumer or from the content provider to service provider. The objects located in the service consumer principal service layer are able to control the delivery and display of the information in the service consumer system without the need to interact with the service provider. DAVIC 1.0 only supports uni-directional audiovisual data from a service provider to the service consumer. This implies that applications that require bi-directional audiovisual flows such as telephony, videoconferencing are not supported by the current DAVIC systems. Future DAVIC specifications should solve this.

DAVIC provides a functional, non-object-oriented specification of their systems. The DAVIC specifications are examined from an ODP computational and engineering viewpoint since these viewpoints fit most closely to the DAVIC 1.0 specifications. The information viewpoint is omitted since DAVIC 1.0 specifications do not contain detailed information models. The author transformed the DAVIC 1.0 specifications to ODP computational and engineering models to enable comparison of the various architectures described in this chapter. The focus is on the service provider and service consumer systems since these two entities primarily deal with multimedia applications.

### 4.4.1 Concepts related to the computational viewpoint

Figure 4.9 shows the three main systems and their external interfaces that are visible from a computational viewpoint. The delivery system is depicted in the computational view as a binding object offering a control interface to manipulate the connection. This

section focuses on the service provider and service consumer and those interfaces that are related to multimedia.

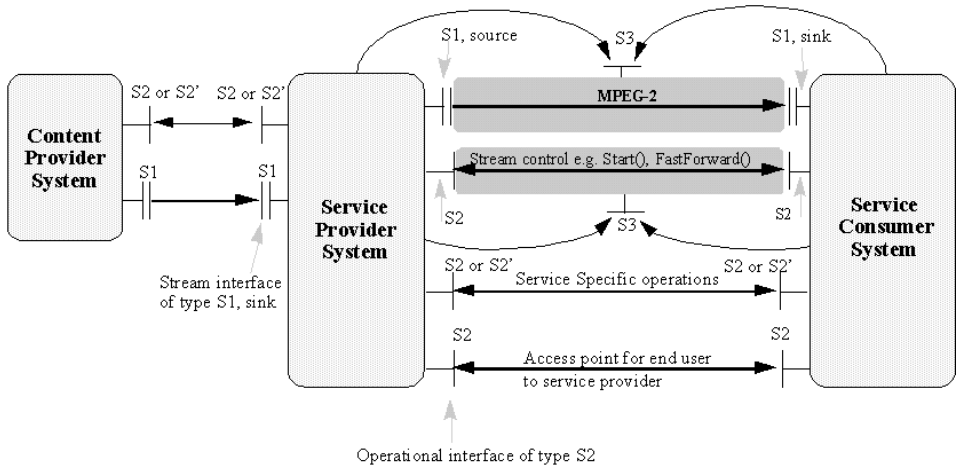



Figure 4.9: Computational view on a DAVIC system

## Multimedia interfaces

In ODP computational terminology, the DAVIC *S1 interface* represents a stream interface of type S1, which is used for the interaction between service provider, content provider and service consumer. S1 type stream interfaces are used to transport high-bandwidth, bulk data, using uni-directional flows from service provider to service consumer. The S1 flow can consist of MPEG1-2 encoded audio/video and associated data, binary objects and other content-information types to be used by the service consumer. It is transparently passed through the delivery system; its content will be interpreted by the service consumer system.

The *S2 interface* is an operation interface used for bi-directional communication between service provider service elements (see service provider below) and service consumer. It enables the service consumer to dynamically control an established service session. The possible operations described by the S2 type interface are service element specific. In the case of a video-on-demand service, the S2 type interface typically describes VCR commands.

The S2 interface has two purposes:

- It is used for the exchange of control-information between an application service element (part of service provider) and the service consumer's set-top box. DAVIC uses the MPEG-2 Digital Storage Media Command & Control User-to-User (DSM-CC-UU) protocol [42] to control a service session. In this case the S2 interface is used for user to application interaction. Operations are defined to control the audiovisual flows (e.g. pause, stop, and fast forward), to alter the rate of the flow (scale-function). Basic operations (start/stop) to control the life cycle of multiple services are also part of this interface;
- It is used to provide a DSM-CC Download Control service (indicated as S2'). In this case, a uni-direction channel exists from service provider to service consumer set-top box to download data (e.g. an executable piece of code). A separate channel exists between service provider and service consumer, which controls the data that is downloaded. This separate channel enables the user to indicate the properties and configuration of the set-top box.

### Multimedia Binding object

Bindings that are established between the service provider and service consumer are explicit bindings in ODP terminology. DAVIC specifications allow both service provider and service consumer to control the established binding through the S3 interface. DAVIC makes it possible to alter the binding for both the S1 interface and S2 interface independently. This is shown in Figure 4.9, using two binding objects.

The *S3 interface* is an operation interface used for session control. A session is a logical entity representing the state of the interaction of the service consumer with the server. A session (e.g. teleshopping session) may require multiple connections (e.g. a video channel and separate channel for dynamic control), and may require dynamic network resource management during the lifetime of a session. The S3 interface is used for this purpose. Operations on the S3 interface can be invoked to alter the configuration of the session that has been established (e.g. service provider adds resources in the delivery system to improve performance).

DAVIC supports point-to-point and point-to-multipoint configurations for its binding object. For S1 interfaces that are bound, the binding is a uni-directional configuration from one source to one or more sinks. S2 interface bindings are always point-to-point configurations, which can be bi-directional.

### Service Provider

This section focuses on the service provider computational objects and its interfaces. The service consumer is not discussed since DAVIC specifications assume that the service consumer system is located in one physical location. On the contrary, the service provider can be distributed over several computing nodes.

The service provider system is defined as a set of distributed *Service Elements* each providing a particular function. A service element can be related to an ODP computational object since a service element provides one or more interfaces and a certain functionality [38]. DAVIC defines a standard service element that serves as a base description for the more specialised service elements as depicted in Figure 4.10. Note that the computational configuration shown in Figure 4.10 does not imply that the service elements need to be located in the same physical system.

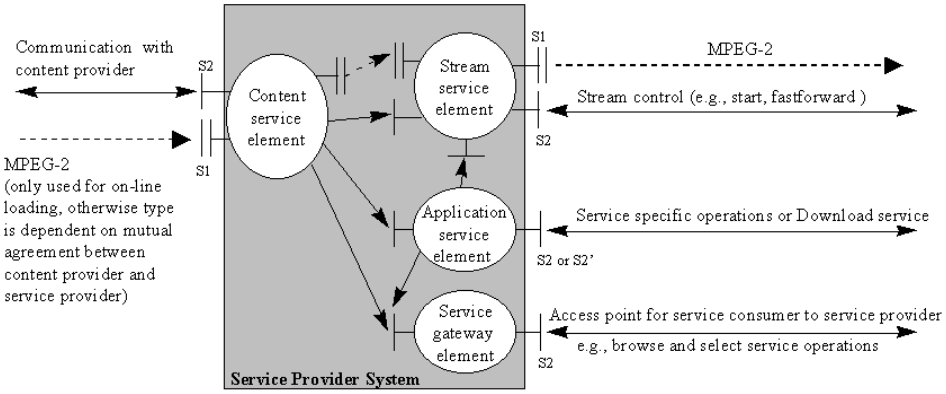



Figure 4.10: A computational view on the interfaces supported by Service Provider system

Figure 4.10 presents a decomposed computational view of the service provider system and shows four core service elements dealing with applications. The service elements provide external interfaces of type S1, S2 or S2', for communication with content providers and service consumers. Internal interfaces are needed for the exchange of information between the service elements themselves. These internal interfaces are implementation specific and not standardised within DAVIC.
The *Content Service Element* is used for interaction with content providers. It is the object where the contents, for example, movies and games are located. The main functions performed by this object are service installation and service management. It interacts with the other service elements, especially with the stream service element, which transports the content to a service consumer. The content service element receives the content from a content provider either on-line through the S1 type stream interface or off-line using other transportation means (e.g. a PAL videotape which is then digitised into MPEG format and stored in the content service element).

The *Stream Service Element* is a both repository and a source for streams. It has a stream interface of type S1 when audiovisual flows or data flows are sent from the stream service element to the service consumer. The stream service element exports an S2 type interface through which clients can control the media stream (e.g. stop, fast forward, resume).

The *Application Service Element* represents a collection of objects, which are the applications supported by the service provider. The S2 type interface supported by the

application service element provides application specific operations or the download capability as explained before.

The *Service Gateway Element* is a broker where services located in the application service element make their existence known, and deregister when they are not available anymore. Through the service gateway, the service consumer is able to discover the existence of a particular service in the service provider system. It is also the means by which service consumers can activate/deactivate an instance of a service. The service gateway exports an S2 type interface through which the service consumer can browse and select services offered by the service provider.

### 4.4.2 Concepts related to the engineering viewpoint

DAVIC 1.0 specifications focus on the possible communication paths between service provider, delivery system and service consumer systems. Except for security functions, they do not specify in detail the functions needed to process the audiovisual data. Figure 4.11 shows an ODP engineering specification of a service consumer system.

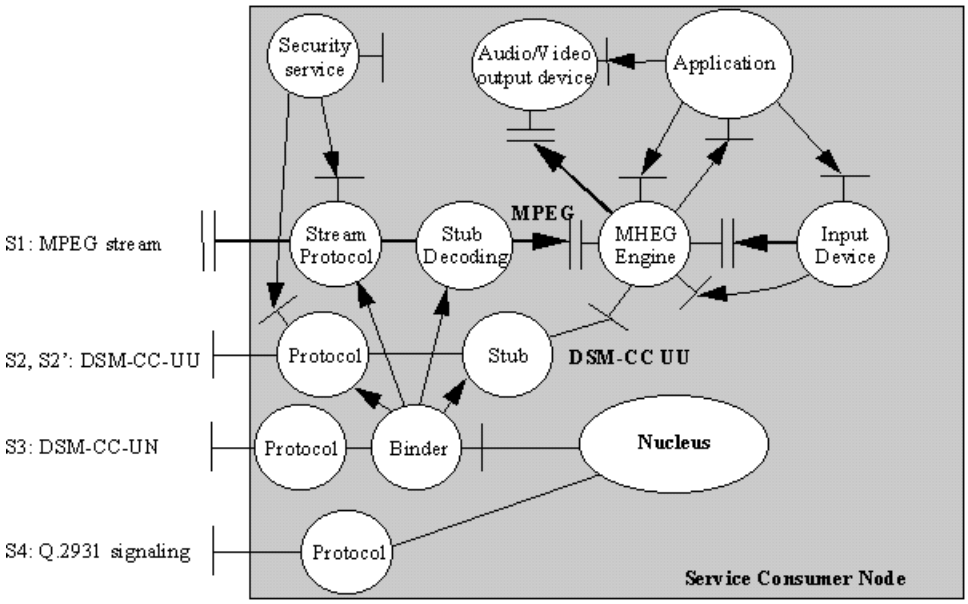



Figure 4.11: Engineering viewpoint[9] of Service Consumer System

DAVIC uses the MHEG-5 standard [80][146] for the support of distributed interactive multimedia applications in a heterogeneous client/server environment. MHEG-5 defines

9 The S4 flow is used for the establishment of a (physical) connection between service providers and service consumers. In ODP terminology it solves the location transparency functionality. Therefore, the S4 flow is only visible in the engineering viewpoint and not in the computational viewpoint.

a final appearance representation for application interchange. MHEG-5 applications specify how audio, video and textual information is organised as it is presented to the end-user. This means that MHEG-5 applications only need to be authored once and can run on any MHEG-5 compliant platform. MHEG-5 defines the syntax and semantics of a set of object classes that can be used for interoperability of applications across minimal-resources platforms. In Figure 4.11 the *Nucleus*, represents a light weight operating system providing the minimal resources (memory, processing) needed by the engineering objects located in the set-top box.

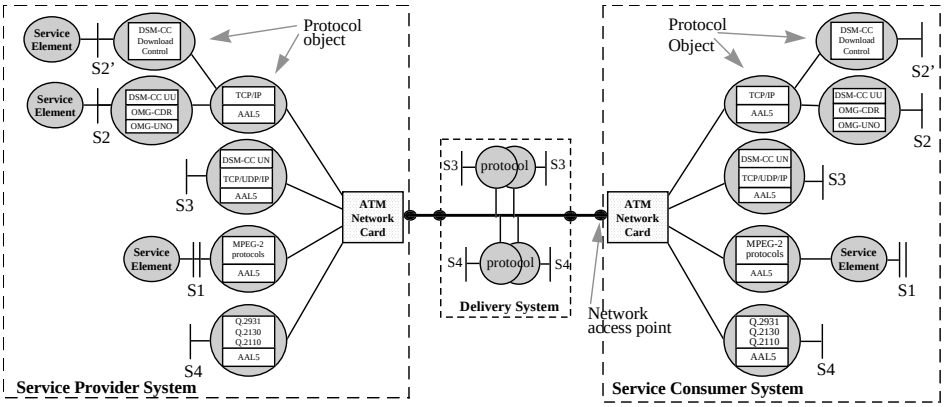
The *MHEG engine* is an important engineering object in the service consumer node. The MHEG-5 engine manages the interpretation and the presentation of MHEG objects under the control of the application using it. Before an MHEG object can be presented to the end user the MHEG-5 engine must prepare it. This consists typically of retrieving it from the server (using the stream protocol object), decoding the interchange format (by the stub decoding object), creating the internal data structures and making the MHEG object available for further processing. An MHEG engine performs syntax analysis, coding, management, interpretation and presentation of MHEG objects.

Typically the MHEG engine receives local input from an *Input device.* The input can be either control information (e.g. VCR commands) or audiovisual input in case of an interactive application.

A *Security service* is available to provide secure data exchange facilities (e.g. encrypting data). The security service is able to encrypt the data sent through the channel objects.

### Channels

A considerable part of the DAVIC 1.0 specifications focuses on the details of the *Channels*. It describes the protocol stacks to be used to convey the S1-S4 flows. Figure 4.12 shows the correspondence between the protocol stacks defined in DAVIC 1.0 for the S1-S4 flows and an ODP compliant engineering viewpoint specification. The S1 stream interface and S2 operation interface convey information, which is used by the service consumer services. These flows are transported transparently through the delivery system.




**Type specification of stream and operational interfaces**
S1: ISO/IEC 11172-2,3 MPEG1, ISO/IEC 13818-1,-2,-3 MPEG2
S2: DSM-CC UU (User-user control interface)
S2': Download service (DSM-CC Download control) ISO/IEC 13818-6
S3: DSM-CC UN (User-network control interface)
S4: B-ISDN signaling interface (Q.2931)

Figure 4.12: Engineering/technology viewpoint on the channel objects

Four different channel types are involved in the exchange of time-based flows from the service provider to the service consumer.

The *S3 interface* enables resource reservation in the delivery system, in the service provider and in the service consumer system. It enables the service consumer and service provider to invoke operations for the establishment of a session (after a basic channel has been established between the service provider and service consumer using the S4 signalling interface) and to allocate resources. DAVIC uses a subset of the operations defined by the DSM-CC User-Network (DSM-CC-UN) protocol. DAVIC 1.0 only supports service consumer initiated session set up. Either the service consumer, the service provider or the delivery system can initiate session teardown. The service provider always initiates addition and release of resources.

The *S2 interface* is implemented using the DSM-CC User-User protocol. As described previously, either a download protocol or an user-to-user interaction protocol can implement the S2 flow. In case of downloading control information, the information is carried in a high bandwidth channel. The S2 interface is used in this case to download audiovisual data or applications (e.g. games) from the service provider into the service consumer set-top box. Besides a high bandwidth channel for downloading, a small bandwidth channel is used to negotiate download parameters and to control the download of information (e.g. send acknowledgements). If the S2 interface is used for user-to-user interaction, the type of operations that can be invoked depends on the service elements to which the S2 interface is associated.

The *S1 interface* conveys MPEG encoded audio/video, associated data and binary objects from the service provider to the service consumer. The information transported through the S1 interface is transparent to the delivery system. DAVIC specifies several protocol stacks for both ATM and non-ATM environments [42].

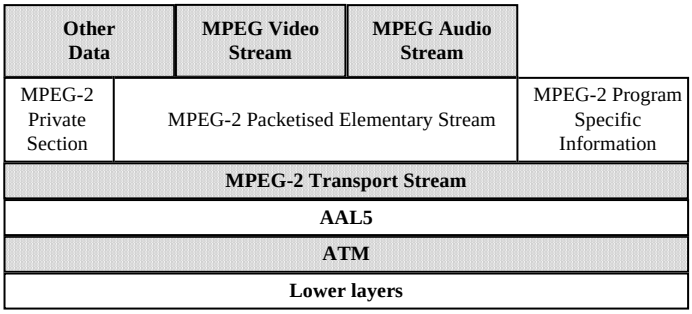



Figure 4.13: MPEG Transport System protocol stacks for ATM

Figure 4.13 shows the protocol stack for ATM based interaction. The elementary MPEG Video, MPEG audio stream and Other Data (such as graphics, files) are packetised in the MPEG-2 Packetised Elementary Stream and transported through the MPEG-2 Transport stream layer (and lower ATM layers) to the other node. The MPEG-2 Private section and MPEG-2 Program Specific information are placeholders for additional information that is to be transported, such as conditional access tables, network information tables etc.

### 4.4.3 Discussion on DAVIC

The architecture proposed by DAVIC 1.0 consists primarily of a 1-way flow of audiovisual data from a service provider to the service consumer, though DAVIC specifications state that bi-directional audiovisual flows should be possible. DAVIC services are targeted for the home environment with minimal equipment available at the service consumer site. High downstream bandwidth is available (e.g. cable) and small upstream bandwidth (e.g. telephone) is needed for interactive control of the audiovisual data. This is unlike the platforms targeted by IMA, where both client and server have equal bandwidth at their disposal.

DAVIC 1.0 Part 2 defines the S1-S5 information flows. The definitions for S1-S5 are re-iterated in Part 7[10], but the author encountered various inconsistencies and confusion. An information flow is carried on protocol stacks, but the protocol stack itself is not the flow: it supports the carriage of the flow. DAVIC should allow different protocol stacks to carry the same S-flow, or equivalent protocol stacks for different S-flows. It should also be possible to use the same instance of a protocol stack to carry more than one S-flow (e.g. S1 and S2 over an MPEG Transport Stream). DAVIC should separate

10 Due to inconsistencies in the various DAVIC specifications is Part 7 leading compared to Part 2 and Part 3 according to DAVIC insiders.

between the interface specifications (i.e. S1-S5 flows) and the protocol stack that carries the information flow. Using an object oriented paradigm as proposed in this section solves the problem of mixing information flows (i.e. the computational interfaces) and the supporting protocols (i.e. engineering channels).

DAVIC specifications do not describe a QoS negotiation mechanism to provide certain guarantees for the data that is exchanged between the service provider and service consumer. DAVIC relies on the MPEG and DSM-CC standards for the specification of QoS. The DSM-CC standard supports the operations to specify QoS for a certain session between the client and server. It is possible to specify network related QoS parameters such as network delay, network jitter, error characteristics and bandwidth availability for the support of both CBR and VBR audio/video. At the application level, the quality of the audio and video flows depends on the parameters set to encode these flows into MPEG1-2 format.

The main multimedia elements in DAVIC specifications are the MHEG-5 and MPEG standards. DAVIC does not focus on the applications and expects industry to build its own multimedia applications using the underlying DAVIC technology. The growing popularity of Java [84] makes the author belief that DAVIC technology should be compatible with or migrate towards solutions that interoperate with JAVA technology. This tendency is already visible in new releases of the DAVIC specifications where for example the MHEG-5 engine can be executed on top of a JAVA virtual machine.

## 4.5 Microsoft Interactive Television

MiTV is a software platform designed for the support of a wide variety of interactive applications running on a broadband network. MiTV expands the scope of broadband services by combining the delivery of existing cable and TV broadcasting with the delivery of interactive services. The initial implementation of MiTV focuses on delivery of interactive television to the home TV and personal computers. In the future, broadband network services will extend to other home and office equipment, such as audio systems, video game boxes and hand-held services.

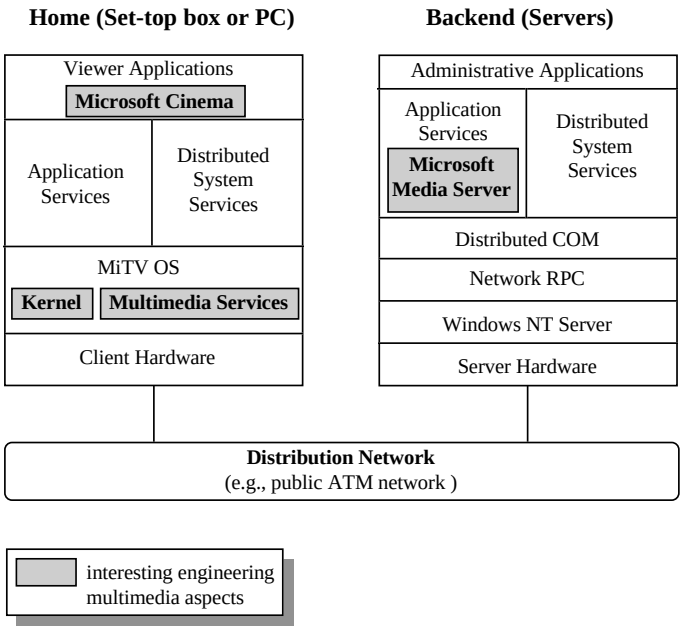



Figure 4.14: MiTV architecture

Figure 4.14 shows the MiTV software architecture that interconnects client set-top boxes with MiTV backend servers. MiTV uses the COM technology. The backend services can be accessed using the distributed DCOM standard of Microsoft.

## Backend server

At the backend, several servers can be interconnected through a LAN to provide a collection of services such as distributed system services, administrative applications and application services for the MiTV system.

*Distributed system services* include the directory service, object store service, class store and security services. The *directory service is* used to locate and access objects throughout the network without knowing where the objects are physically located. This service runs on all the MiTV clients and all servers that store objects. The *object store service* stores the body of stored objects that are available to MiTV applications. Objects stored could be data objects (e.g. subscriber information) and service objects (e.g. a logging service). The *class store* service is a repository for all executables and DLLs (e.g. icons and fonts) used by MiTV clients. These executables and DLLs are downloaded by the client and executed locally. The class store also provides the source of the set-top box's operating system. The *security service* implements a variety of security techniques including public key encryption, symmetric key encryption and one-way hashing. These techniques support features such as mutual authentication, secure payment, and digital signatures that are required for secure communication on the network.

*Backend application services* include the Microsoft Media Server, database services, interactive subscriber services and financial transaction services. The *Microsoft Media Server* is a file storage and delivery system where digitised data is stored and can be accessed on demand by MiTV clients. *Database services* enable MiTV applications to access data. The backend contains a number of databases used by the MiTV applications to store information about television programming, videos stored etc. *Interactive Subscriber Services* (ISS) provide applications running on the set-top box and at the backend with subscriber information. Information typically stored in ISS consists of subscriber profiles, viewer profiles, and information about the equipment on the subscriber's premise. Financial Transaction services (FTS) provide the MiTV system with a secure and uniform method of conducting financial transactions. FTS provides a framework for pricing, taxing and receiving payment of goods that are sold through services offered by the MiTV system.

*Administrative applications* include a range of tools that enable network operators to manage the MiTV system. Tools are included to manage the network and the services running on the backend servers and MiTV specific applications and information used by the system.

### Client set-top box

The *set-top box*[11] is the primary hardware component at the client side (located in a home). The set-top box is comparable to a cable decoder box. It also includes the computing and network hardware required for handling interactive digital channels. The set-top box is a general purpose programmable computing device, which is capable of processing digital media in a variety of forms including audio, video and animated graphics. The set-top box includes MPEG decoding capability to convert digital signals into standard video signals such as PAL, NTSC or SECAM that can be received on regular televisions. The viewer interacts with the set-top box using a hand-held remote control.

The set-top box ROM contains only minimum boot code. The boot code downloads the MiTV operating system (MiTV/OS) and all applications that are needed. The MiTV/OS is a compact, high performance operating system, which supports multimedia features. The services located in the set-top box are the MiTV/OS, distributed system services (see above), application services and the viewer applications.

The *MiTV/OS* includes the kernel, direct draw, multimedia services, a subset of Windows-32, and TV-User Interface Controls. The *kernel* is designed for high performance in real-time and is capable of interpreting multiple digital media streams simultaneously. The kernel provides for the management of processes, memory and objects. It is the nucleus in ODP terminology. *Direct Draw* is a high performance

11 The set-top box is a personal computer in the MiTV version available at KPN Research. In this case, the PC emulates the functionality of a set-top box and it is extended with specialised graphics hardware and network adapter cards.

graphics engine that provides direct access to hardware graphics. It provides the functions to manipulate different types of graphics (e.g. bitmaps). *Multimedia services* provide the co-ordination and synchronisation functionality of time-based flows. Multimedia services support the functionality to build and manipulate displays. It shields the application designer from low-level functions to control and manipulate time-based flows (see also below). The *TV user interface control* service manages the presentation and the use of interactive TV user interface. It provides control functions that work with a normal remote control for television sets.

The *Viewer applications* include the following services: the MiTV Channel manager, MiTV Customer Service Application, Microsoft guide and Microsoft Cinema. These are specific applications to navigate through the applications and programs that are offered by the backend server. An interesting application is Microsoft Cinema, which is an interactive movies-in-demand service (see below).

#### Distribution Network

MiTV is based on a switched distribution network that provides two-way communication. Unlike the DAVIC 1.0 specifications, high bandwidth channels are available in MiTV for both directions. MiTV uses the Internet Protocol (IP) as the network layer and ATM for the data link layer. Several solutions are possible for the physical layer such as hybrid fiber coax (HFC), fiber to the curb (FTTC) and fiber to the home (FTTH).

### 4.5.1 MiTV related to the computational viewpoint

The MiTV documentation [141][142] describes the functionality and user interfaces provided by the various applications that constitute the MiTV system. Internal models are not described, Microsoft only provides APIs to the application developers to interact with an MiTV application. An MiTV application can be regarded as a large computational object supporting one or more interfaces, and encapsulating the functionality provided by that application. The interfaces are based on the COM model, which does not completely conform to the OO principles applied in ODP-RM (see also discussion in Section 3.4). To enable comparison between the architectures described in this chapter the following applications, which are of interest for multimedia are positioned in the ODP computational viewpoint.

#### Microsoft Cinema

Microsoft Cinema is an interactive service for requesting movies. The viewer can control the playback of movies using VCR like commands such as pause, play, stop fast forward. The Microsoft media server running at the backend server services Microsoft Cinema. Besides the Cinema application, Microsoft provides an extensive set of content authoring and application development tools (e.g. MiTV Movies on Demand (MOD) application).

**Microsoft Media Server (MMS)**
MMS is a file storage and delivery system for the access to stored digitised data. MMS locates the requested file and transmits it at a guaranteed and constant data rate to the set-top box. MMS stores the videos used for by video-on-demand applications such as Microsoft Cinema. MMS is a distributed application, which can be physically located on one or more networked computers. It can serve many users at the same time.

### 4.5.2 MiTV related to the engineering viewpoint

In this section, the focus is on several engineering functions dealing with flows. Microsoft documentation does not provide internal details but only describes the functionality provided by these services.

**Kernel service**
The kernel service manages multimedia objects and is capable of interpreting multiple digital media streams simultaneously. It fits entirely in the set-top box ROM and RAM. The kernel service provides functions for *real-time scheduling.* Applications can use scheduling constraints (a set of scheduling parameters to be applied to the execution of a block of code) to achieve fine-grained control of real-time behaviour. *Constraint objects* can be attached to threads for the purpose of keeping track of time and criticality. A constraint object contains the following values: start time, estimated time and deadline.

Real-time applications often need to reserve resources to operate properly. The Kernel provides *resource management* functions to manage system resources (e.g. memory, CPU) and guards the agreements made with the end-user. Examples: a video-phone call should pause a movie unless it is being recorded; a video should be degraded before audio when the set-top box gets overloaded.

**Multimedia data stream synchronisation service**
The Multimedia service provides for the co-ordination and synchronisation of audiovisual streams in the set-top box. Multimedia services include a set of objects for building and manipulating displays. These objects shield application developers from low-level components and can be called by any process or application, which needs to control an audiovisual stream. In this section, the author focuses on a particular service (synchronisation) which is of interest for Chapter 6.

*Filters* are the primary objects of the synchronisation system. They model the media processing elements in the system (e.g. a video or audio device). Filters are modelled as COM objects. Depending on the type of filter, it has several *pins* (pin is synonym for port). A pin represents a data flow that can be either incoming or outgoing. A pin is strongly typed (e.g. an MPEG1 flow) and MiTV allows pins to support multiple types, which can be selected by the user (e.g. pin supporting MPEG1 or MPEG2 flows). Figure 4.15 shows the relation with ODP-RM terminology. A filter represents an

engineering object abstracting from a particular device present in the system. It supports one or more pins, which is similar to an engineering object having several input/output flows. The filter can be controlled through an interface, which is similar to an ODP operation interface for controlling time-based flows.

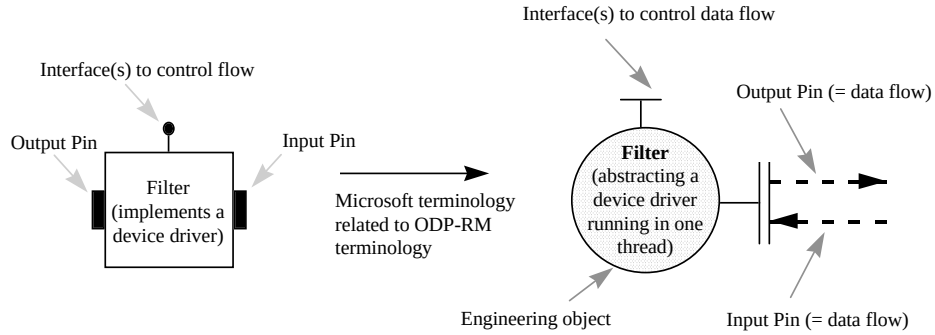



Figure 4.15: MiTV terminology related to ODP terminology

### Binding object for synchronisation

MiTV distinguishes between three different filter objects. *Source filter* objects generate data (data from an input device is captured in the source filter). *Transform filter* objects are able to modify data streams, and *Renderer filter* objects are responsible for the representation at the other end.

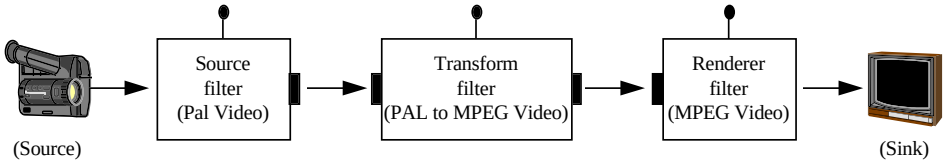



Figure 4.16: Multimedia objects (simple filter graph)

These filter objects are grouped in a so-called `SyncGroup` for the purpose of synchronisation time-based flows. The pins of the filter objects are linked to each other to form a dataflow path from source to sink (see Figure 4.16). The dataflow is modelled as a graph of related filters the behaviour of which needs to be co-ordinated in time. The `SyncGroup` can be regarded as a particular binding object in ODP terminology. It provides a control interface for the manipulation of the filter objects part of the `SyncGroup` (Figure 4.17, ①). The `SyncGroup` provides synchronisation and filter management services. Operations to manipulate a `SyncGroup` are adding or removing filter objects, and setting the reference clock. Also the propagation delay for the `SyncGroup` can be determined. Propagation delay is the maximum amount of time that a sample (audio sample, video frame) needs to go through the longest path through the graph. Like IMA and TINA, MiTV also uses a graph model to describe the relations

between the filter objects involved in a single `SyncGroup`. All filters that need to co-ordinate their behaviour in time must be member of exactly one `SyncGroup`.

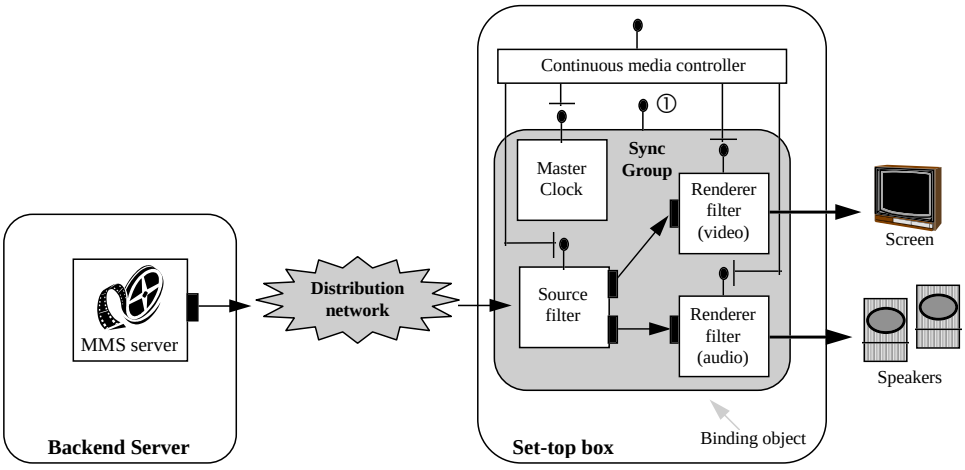



Figure 4.17: Video player example

A continuous media controller (e.g. VCR remote control) controls the SyncGroup. The MiTV object model (COM) supports the aggregation principle. This means that an OLE object (SyncGroup) encapsulates the services of other objects (filters). When a client (continuous media controller) invokes an operation on the encapsulated component (SynchGroup) it can in turn invoke an operation on the contained component (filter).

MiTV supports both inter- and intra-stream synchronisation. *Inter-stream* synchronisation is defined in MiTV as synchronising two streams with respect to each other. *Intra-stream* synchronisation is defined as the sequencing of samples within a stream[12]. To achieve synchronisation, time plays an important role. MiTV defines three different time systems that are used for synchronisation. The *reference time* is determined using a typical time source such as an oscillating crystal, which is a monotonously increasing time source. The reference time is used as the basis and everything else is measured against it. The reference time is the same for all the filters in the same `SyncGroup`. The reference time is the master clock for a `SyncGroup` (see Figure 4.17). *Presentation time* is the time at which samples need to be presented to the end-user. *Logical time* is the time relative to the stream origin, it is the user's perspective on time and is analogous to a position in a stream.

12 The definitions of intra-stream and inter-stream synchronisation given in MiTV documentation are opposite compared to those in other literature. The author applies the 'normal' use of the concepts inter-stream and intra-stream synchronisation, and choose to alter the MiTV definitions to avoid confusion with Chapter 6. See also Chapter 6 for a more precise definition.

Using the different time systems as described above, intra-stream synchronisation is achieved in MiTV by timestamping all incoming samples with a presentation time and by using the reference time to determine the rate of presentation.

Inter-stream synchronisation is more complicated. The master clock timestamps all samples with their presentation time. This occurs in two places: in the backend server when the samples are encoded and in the set-top box (source filter). The source filter in the set-top box takes the presentation time that is stored in the sample, and gets the current time from the Master Clock object, to which it adds the propagation delay from the `SyncGroup`. The result is a new presentation time, which indicates when the sample should be displayed. The rendering filter of the same `SyncGroup` receives this sample, and it gets the reference time from the Master Clock. The renderer filter compares the reference time to the presentation time stamp of the sample and then decides to playout the sample or not. In case of synchronisation problems (the presentation time and reference time differ too much) the renderer filter can apply a particular mechanism (e.g. skip the sample or pause it). MiTV defines several mechanisms for coping with synchronisation problems.

### 4.5.3 Discussion on MiTV

The MiTV documentation provides a functional description of the various applications and how to control them. The architecture of MiTV is straightforward and functionality is dispatched between the backend server and set-top box. MiTV applications are to a great extent proprietary implementations and only allow independent software vendors (ISV) to add other applications that conform to the interfaces prescribed by MiTV applications.

The MiTV concept of `SyncGroup` shows similarities to the Virtual Connection object in IMA. MiTV has chosen a particular solution to synchronise multiple time-based flows, an issue that is left open in IMA. It should be noted that MiTV does not reference IMA documentation.

MiTV is based on ATM as the underlying network and an ATM to the Home (ATTH) solution is promoted. ATTH provides advantages such as end-to-end QoS guarantees and bandwidth guarantee. Different connection configurations as supported by ATM (point-to-point up to multipoint-to-multipoint connectivity) are advantageous for the various types of services that will be used in the home environment. However, it is unlikely that ATM will be generally available to the home environment within the next few years [144]. It is expected that the tremendous momentum of Internet, which does not depend on ATM, will force Microsoft to come up with multimedia services that also operate in a non-ATM environment [145].

The MiTV set-top box is currently only implemented on a PC platform. No dedicated set-top boxes are yet available. The technology has to compete with JAVA solutions but

it is expected that Microsoft is able to adapt the MiTV system so that JAVA applets can be downloaded in the set-top box.

Jim Green (Microsoft's group program manager, responsible for the MiTV project) said: *"MiTV is THE premier, platform independent, fully-distributed, object oriented, real-time, multimedia operating system for the 90's and beyond."* It is the author's view that MiTV-OS is neither really open (except for its COM interfaces and the supporting ATM network) nor purely object-oriented (except for its interfaces). Although it constitutes a nice distributed multimedia operating system, and Microsoft has developed many applications to put on top of it, the solutions are Microsoft specific. Microsoft might win the short-term market race but with respect to openness, MiTV is not the solution for the medium and long term. It is a proprietary solution and migration will consist of replacing Microsoft applications with new ones. Microsoft defines the most important applications of MiTV and ISVs need to adapt their applications to the MiTV applications. Openness is in this case to be dependent on Microsoft.

## 4.6 Conclusions

Both IMA and MiTV abstract from the supporting network and are based on standardised protocols. Their models start from this point onwards and most intelligence is put in the peripherals attached to the network. This reflects the situation that both MiTV and IMA are initiatives from the computer industry, which have little influence on the network. They assume certain QoS guarantees from the network. On the other hand, TINA is a telecommunications driven initiative and their solution is to put as much as possible intelligence into the network (see also Figure 4.4). The author believes that TINA goes too far stating that the CSM object (an object in the domain of a network operator) should have insight in the end-user node and responsible for the set up of an internal node connection. It should be noted, however, that the MiTV backend server can also get insight into the configuration of devices in the set-top box. But this is from a service provider (and product) position, which is more reasonable than TINA's solution (that should be generic enough and independent of the configuration of the end-user node).

TINA mainly focuses on the generic parts of service control and connectivity whereas other consortia have developed more detailed specifications. It is expected that OMG CORBA will be used as an implementation of the TINA-DPE [72]. But CORBA does not yet support the concepts of streams and stream interfaces, although activities within OMG have been started to investigate to which extent and how this could be done. For TINA specifications, this implies that a gap exists between the information/ computational modelling of multimedia applications and the actual implementation on a TINA-DPE.

DAVIC provides the technical infrastructure to transport multimedia data from service provider to service consumer and specifies in detail which protocol stacks should be used for the various information flows (S1-S5 flows). DAVIC 1.0 does not specify the

services itself. On the contrary, TINA does not describe in detail the underlying infrastructure, which is needed to exchange data between different stakeholders. TINA depends on the results of OMG and ATM forums.

TINA has not yet defined the service specific control operations, nor stream interfaces for different classes of multimedia applications. The focus has instead been put on generic aspects of service control as expressed in the generic service session concepts. The author regards this focus on generic aspects as a strong point since other consortia, e.g. DAVIC and IMA, already work on these kind of specifications. For example, specific control operations for video-on-demand services are defined in DAVIC. Furthermore, wide acceptance of generic service session concepts that form the basis for many kinds of multimedia services will allow greater interoperability between them. There is however still a need for alignment between the work done in TINA and 'service specific' consortia.

Both DAVIC and Microsoft MiTV try to relate their specifications to Internet technology. The developments in this area go fast. At the time of writing this thesis, the Microsoft MiTV project has been cancelled although many parts are being re-used in other Microsoft products that are closely aligned with the Internet technology.

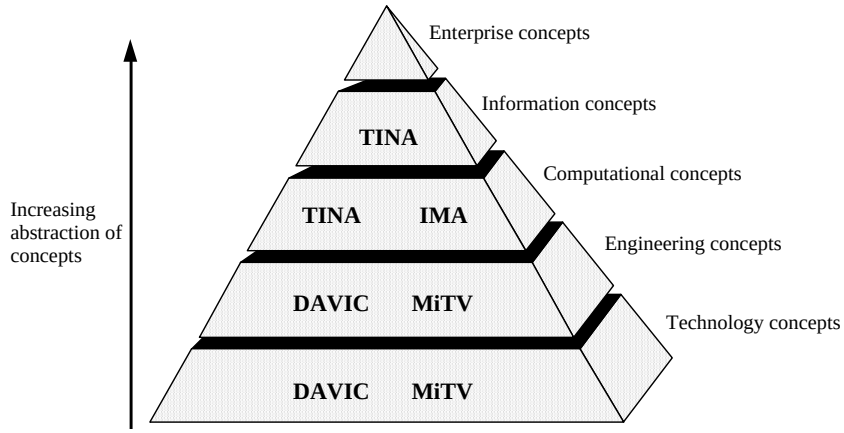


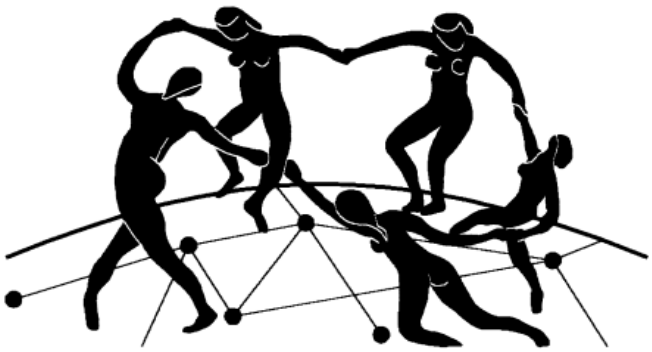

Figure 4.18: Consortia positioned along the ODP-RM viewpoint languages

The concepts used in various consortia are positioned in the ODP viewpoints and corresponding viewpoint languages (Figure 4.18). TINA focuses on high level concepts that have no correspondence to specific engineering and technology implementations. IMA defines several concepts that relate to the ODP computational and engineering viewpoint. DAVIC and MiTV both focus primarily on the engineering and technology issues. Their concepts are suitable for a specific technology solution where MiTV concepts are the most proprietary ones, which can not easily be used outside a Microsoft environment.

Difficulty was encountered in positioning consortia concepts in the ODP viewpoints due to the fact that each consortia uses its own abstraction principles that do not correspond to the ODP-RM viewpoints. TINA is an exception, although it does not make a proper distinction between the computational and engineering viewpoint.

The author concludes that the various consortia define their own (specific) solutions and concepts. Especially, in the area of specific technology the solutions differ considerably. At a higher abstraction level (computational and engineering) similarity exists. Only at this level, common terminology is useful. This means that at computational and engineering viewpoint multimedia concepts should be standardised (providing interface descriptions etc.). Each consortia should then do a particular mapping onto standards and specific technology. Technology evolves so fast that it is unrealistic and even undesirable to have a single technology mapping.

A widely accepted framework is needed to relate and combine ongoing work and existing solutions. Concepts should be described at a sufficiently high abstraction level. Rules should define how such a specification can be related to specific technology. In Chapter 5 the ODP-RM computational and engineering multimedia concepts are refined. This results in a specification for an open DPE.



# 5 ODP Computational and Engineering View on Distributed Multimedia Processing Environments

*This chapter applies the ODP-RM computational and engineering views on the specification of and support for distributed multimedia services. In the computational language, the ODP concepts of 'stream interface', 'environment contract', 'binding object' and 'binding action' necessary to specify the multimedia concepts of a distributed application are refined. Then the correspondence between these computational concepts and engineering concepts such as 'stream interface reference', 'engineering channel' and 'engineering channel establishment' are described. Further, an example is included that illustrates the correspondence between an engineering configuration and a specific technology solution.*

## 5.1 Introduction

Real-time multimedia services have additional functional and performance requirements, which need to be met by DPEs. They require for example support for a variety of protocol stacks suitable for stream communication. In addition, support for the negotiation and manipulation of QoS parameters is needed to provide performance guarantees. The underlying infrastructure should support strict performance

requirements imposed by real-time multimedia applications. Another important issue is interoperability between multimedia services located in different computing nodes or even in different administrative domains. This means that multimedia characteristics should be specified in a standardised manner, which is hardware and software independent.

From the discussion in Chapter 3, the need for an overall architecture, such as ODP-RM, is identified to align the different solutions proposed by various consortia and standards. The objective of this chapter is to refine the ODP-RM computational and engineering concepts as presented in Chapter 2 and to be able to specify the time-based media concepts for multimedia services.

As described in Chapter 2, computational objects that deal with time-based media support stream interfaces have a different interaction semantic than operation interfaces. Stream interfaces are of 'type' producer/consumer because time-based communication is not represented as a sequence of invocations due to its potentially unbounded nature [106]. This requires stream interfaces need to be bound explicitly [16]. So-called binding objects are introduced which are capable to relate two or more stream interfaces. A binding object abstracts from distribution and the underlying network technology to support multimedia exchange. This chapter focuses on a class of binding objects that is used for the exchange of time-based media. The author also proposes extensions to the current OMG-IDL for specification of a stream interface and its characteristics.

Based on the ODP engineering model for operation interfaces, an engineering configuration is proposed which is more adapted to the support of computational stream interfaces and associated QoS specification. The ODP 'interface reference' description is used as a basis and refined to incorporate QoS specifications. Channels are needed for the transport of time-based data. Channel establishment is then described according to the QoS phases defined in the ISO-QoS framework [81].

The generic nature of the ODP-RM computational and engineering concepts enables specialisation of the author's proposals for both TINA and CORBA architectures with respect to streams and binding object, as outlined in [66][88].

This chapter is structured as follows: Section 5.2 refines the computational concepts related to stream interface. Section 5.3 describes engineering concepts and objects that are needed to support the computational specification. This section relates the computational and engineering concepts and gives a detailed view on the stream interface reference and the channel establishment. Section 5.4 shows an example to relate an engineering specification to a specific technology (i.e. ATM). Section 5.5 concludes with a discussion of the on-going standardisation activities to include the proposals described in this chapter.

## 5.2 Computational concepts for multimedia services

This section focuses on the computational viewpoint language, and refines several ODP concepts related to time-based media as illustrated in Figure 5.1. Each object is described by its interfaces, its behaviour and its environment contract (see Section 2.3.3.2). Note that both objects and interfaces have a certain behaviour. ODP-RM defines *stream interfaces* that are used by computational objects, which exchange time-based media.

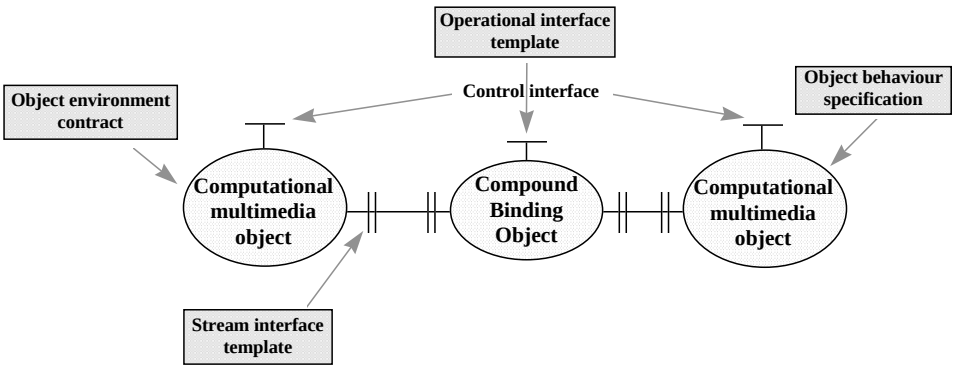



Figure 5.1: Object specification, stream interface and binding object

The compound binding object is used to bind stream interfaces. It represents an abstraction of mechanisms to manipulate the flows that need to be bound. It abstracts from mechanisms such as multiplexers, video bridges and media converters. Additionally, the computational multimedia objects and compound binding object provide control interfaces, which are used for control and management of the flows, and managing the end-to-end binding.

QoS plays an important role in the computational viewpoint for stream interfaces so as to achieve the desirable behaviour of the multimedia service. QoS can be expressed in this viewpoint both statically and dynamically. *Static QoS* means that one wants to define certain QoS properties for the multimedia service. *Dynamic QoS* support implies that it might be desirable to re-negotiate the QoS for a multimedia service during the lifetime of a service session. QoS properties are specified in a static way by means of *environment contracts*. The environment contract expresses the requirements of the object on the environment, as well as, the obligations the object has to fulfil towards its environment [13][16]. The object environment contract is defined for the object as a whole, including all the interfaces it supports. The interface environment contract is specific to each interface. QoS is handled in a dynamic way by a compound binding. Facilities for dynamic QoS negotiation are provided by a control interface of the computational binding object.

Figure 5.1 highlights and relates the computational concepts of concern for multimedia specifications that are detailed in this section. Section 5.2.1 discusses the stream interface concept in detail and provides TINA-ODL descriptions of stream interfaces that can be used by application programmers. Rules for typing and binding stream interfaces are then discussed in Section 5.2.2. These aspects are important for the multimedia compound binding object discussed in Section 5.2.3.

### 5.2.1 Stream interface template

This section details each aspect of the stream interface as it is expressed in the stream interface template. The *stream interface template* includes a *stream signature*, which contains the set of typed flows. For each flow an indication of *causality* is defined, which can be either classified as producer or consumer but not both. Furthermore, the interface template contains the *behaviour* specification of the stream interface, and an *interface environment contract* specification. The computational stream interface environment contract describes, in particular, the QoS characteristics expected from and offered to the environment.

#### Stream signature in TINA-ODL

The stream signature template can be formally described using an interface definition language. The TINA Object Definition Language (TINA-ODL) is used to express the syntax for stream signatures. TINA-ODL has been applied successfully in TINA-C and auxiliary TINA projects. The author believes that TINA-ODL is a suitable language for the ODP-RM computational model.

While defining TINA-ODL, the author introduced a different syntax for stream signatures compared to operation signatures to express clearly the difference in interaction paradigm between stream interfaces and operation interfaces [69]. Use of OMG-IDL for stream interface is possible but OMG-IDL has been designed for operation interfaces only where interactions are expressed as a sequence of invocations. This is not appropriate for stream interfaces (see Section 2.3.3.1). The proposed syntax for stream interfaces allows for the complete specification of time-based flows, specifying their media characteristics including QoS descriptions. The Backus Naur Form (BNF) notation to specify stream signatures in TINA-ODL consists of additional keywords and syntax rules. In [69] the TINA-ODL syntax is detailed.

| | |
|---|---|
| <stream_sig_defns> | ::= { <stream_flow_defn> ';' }+ |
| <stream_flow_defn> | ::= <flow_direction> <flow_type> <identifier> ['**with**' <environment_attribute>] |
| <flow_direction> | ::= '**source**' \| '**sink**' |
| <flow_type> | ::= <type_spec> |

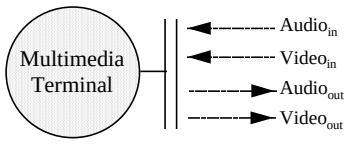
Table 5.1: BNF syntax specification for stream signatures

The syntax defined for the stream signature as shown in Table 5.1 presupposes a causality (i.e. source or sink) of the flow. In the case of a stream interface with bi-directional flows, the choice of client and server may appear rather arbitrary. This

means that the client/server distinction is much weaker for stream interfaces than for operation interfaces. Therefore, it is better to speak of a producer/consumer relation for each flow involved in the binding between the computational objects supporting stream interfaces.

The producer/consumer relation generally implies that a stream interface published by a producer (respectively consumer) object cannot be imported by the consumer (respectively producer) object, as it is the case for client/server related operation interfaces. This is due to the implicit causality in the stream signature. An exception is the case where the stream signature of both the producer and consumer are exactly symmetrical. For example, if both the producer and consumer stream object support $audio_{in}$ and $audio_{out}$, and if the audio flows are of the same type, then one stream signature would suffice. However, if the $audio_{in}$ and $audio_{out}$ types differ, the symmetry is lost and two interface signatures are required to explicitly address the producer or consumer roles of the flows.

With operation interfaces, a client imports a server operation signature, creates a stub dynamically, and the client is able to invoke operations on the server's operation interface. Creating stubs dynamically for stream interfaces is in general not possible due to the specificity of the stream interface, which highly depends on the supporting mechanisms. This is the reason why stream interfaces must always be explicitly bound before communication can take place.




**Stream signature specification**

| | | | | |
|---|---|---|---|---|
| **sink** | VideoFlowType | display | **with** | Video_QoS requiredQoS; |
| **source** | VideoFlowType | camera | **with** | Video_QoS offeredQoS; |
| **sink** | AudioFlowType | speaker | **with** | Audio_QoS requiredQoS; |
| **source** | AudioFlowType | microphone | **with** | Audio_QoS offeredQoS; |

Figure 5.2: Stream signature specification in TINA-ODL

Figure 5.2 depicts a multimedia terminal object offering a stream interface supporting $video_{in/out}$ and $audio_{in/out}$. For each flow in the interface it distinguishes between *offeredQoS* and *requiredQoS* contract depending on the flow causality since the QoS can be different for each flow. The QoS contract is detailed later.

A flow type declaration specifies the characteristics of a flow, for example its coding format or frame size. The flow specification in Table 5.2 shows a set of attributes that are commonly used in the literature [33][89] to characterise audio and video flows. The values for each attribute may change by human manipulation or as a result of changes in the network or end-system. However, not all values of flow attributes are allowed to

change during the exchange of flows. For example, the coding type is fixed when the flow only supports a single coding mechanism (e.g. MPEG).

```
typedef float Compression;  /* compression ratio for audio and video. Range 1:1 up to 50:1 */

/* colour resolution in bits/pixel ranging from grey scale to 65.536 possible colours */
enum ColourType{B/W, 8 bits, 16 bits, 24 bits, RGB, ...};

enum CodingType{NoCoding, JPEG, PCM, CATV, NTSC, PAL, RGB, µLaw, aLaw, ...};

struct Dimension { /* dimension of images */
   float      Width;      /* < 720 pixels, Video signal MPEG coded */
   float      Height;     /* < 576 pixels, Vertical size */
};

struct AudioFlowType { /* structure for an audio flow */
   float             sample_size; /* From 8-bit (telephone quality) to 16-bit (CD quality) */
   float             sample_rate;  /* From 8Khz (telephone quality) to 44.1Khz (CD quality) */
   CodingType        coding;
   Compression       comp_ratio;
};

struct VideoFlowType  { /* structure for a video flow */
   Dimension         frame_size;
   float             frame_rate;       /* Range 25 fps (PAL quality) to 60 fps (HDTV quality) */
   CodingType        coding;
   Compression       comp_ratio;
   ColourType        colour;
};

/* Composition can be applied to specify more complex flowtypes such as a TV flow which is composed of
an audioflowtype and videoflowtype */

struct Tvflowtype { /* structure of a TV signal */
   AudioFlowType  audioflowtype;
   VideoFlowType  videoflowtype;
};
```

Table 5.2: Example of flow type specification for audio, video and TV flows in TINA-ODL

### Stream behaviour specification

The behaviour of a stream interface is strongly related to the flows it encompasses. This is similar for operation interfaces where the operations determine to a great extent the behaviour of an object. Using this correspondence, the type description of each flow has to be investigated. The flow type specification indicates the traffic characteristic for a particular time-period, or in other words, the flow behaviour. The behaviour of a time-based flow can be classified into two traffic patterns: *periodic* and *bursty*. A periodic traffic pattern means that the data units of a time-based flow are generated at regular time intervals. If these data units are fixed in size it is a constant bit rate (CBR) flow. Otherwise it is a variable bit rate (VBR) flow. Compressed video flow is an example of periodic traffic, which can be CBR or VBR. A bursty traffic pattern is characterised by

data units of arbitrary length and generated at random times (i.e. separated by gaps of silence of random duration). Image browsing is an example of a bursty application because the user transfers the images on demand, and such requests are unpredictable.

The behaviour description of time-based flows is strongly related to the stream interface environment contract (i.e. the offeredQoS and requiredQoS descriptions). To support the flow traffic characteristics guarantees must be provided on the throughput, jitter, delay and error QoS parameters.

Concluding, the behaviour of stream interfaces can be related to the individual flows that constitute the stream interface. The time-based flow is instantiated for a certain time-period with a certain traffic pattern, which is called the behaviour of the flow in the context of this thesis.

### Computational object environment contract

In ODP-RM, the *environment contract* defines and specifies specific values or value ranges for the QoS attributes of a computational object and its interfaces. The values in an environment contract are specified between the object and its environment. It expresses the requirements and obligations, which have to be fulfilled, and addresses performance, availability, security aspects, etc. for the objects.

The computational QoS specification describes the requirements related to computational object interaction and is expressed in terms of media characteristics and media relations. *Media characteristics* include source/sink characteristics such as media data unit rate (e.g. 25 frames/s) and transmission characteristics such as end-to-end delay [89]. *Media relations* describe the relations between flows such as the synchronisation skew between an audio and video flow.

In the computational viewpoint QoS parameters are described declaratively in the environment contract. The computational QoS parameter-value tuples are closely related to the parameters defined for the stream interface signatures as illustrated in Table 5.2.

Figure 5.3 shows the various environment contracts for both the multimedia objects and the compound binding object. This section focuses on those QoS aspects that are of concern for the quality of service of time-based flows and ignore QoS issues, such as availability and security here.

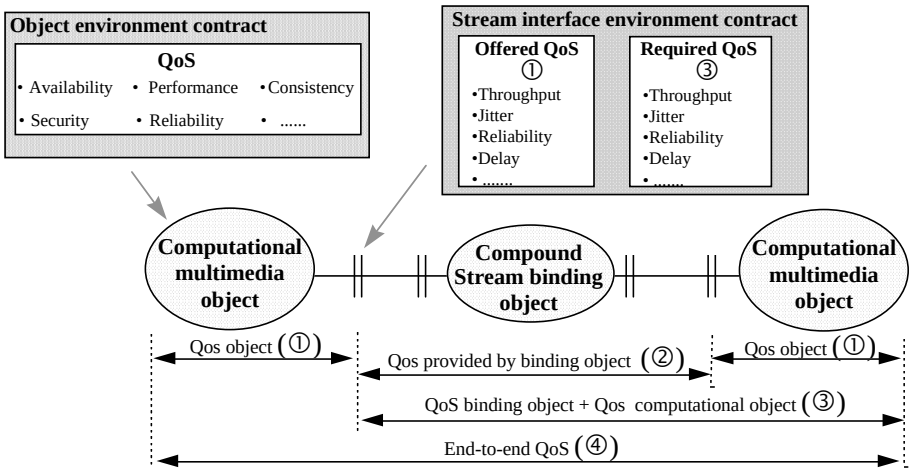



Figure 5.3: Interface and object environment contract for the specification of QoS

Specific object environment contracts can be determined for each computational object. For each interface a contract is specified (Figure 5.3, stream interface environment contract). For stream interfaces, distinction is made between *OfferedQoS* contract for source flows, and *RequiredQoS* contract for sink flows because their QoS values might be different.

QoS specifications in the various environment contracts contribute to the end-to-end QoS that is desired for the multimedia objects involved, as illustrated in Figure 5.3. The end-to-end QoS (Figure 5.3, ④) is composed of the QoS provided by the multimedia objects (①) and the QoS provided by the compound stream binding object (②). The QoS offered by a computational object is determined during processing and (internal) transportation inside the computational object. The parameter values are expressed in the *QoS offered contract* for each flow involved in the binding. The *required QoS contract* is similar to the offered QoS contract but differs for the actual parameter values. In general, the values in the QoS offered contract are 'better' than the corresponding values in the required QoS contract. This is because of the processing and transportation overhead introduced by the other multimedia object and binding object respectively which have a negative effect on the parameter values (③). The QoS specified for the compound binding object specifies values experienced during processing and transportation of time-based media by the binding object (②) that binds the two stream interfaces.

The environment contract specification for the stream interfaces in the binding object should correspond to the environment contract specification for the bounded stream interfaces. For example, a stream offers a throughput of 25 fps and requires 20 fps. The corresponding binding environment contract for this stream interface must state that it offers 20-25 fps and requires at least 25 fps for its throughput parameter.
QoS contract specifications for each interface enables flexible QoS negotiation between computational interfaces. Individual QoS contracts can be established to meet

individual computational object requirements. However, if the established QoS agreements differ between the computational objects, the compound binding object must match these differences. This would require complicate engineering mechanisms and is not always feasible. For example, if a producer sends video frames at 25 fps and the consumer only accepts 20 fps the binding object needs additional mechanisms to match the difference in framerate.

### Relation between environment contract parameters and QoS framework

The ISO-QoS framework provides a good source of information for a widely accepted set of QoS parameters for environment contract [54][81]. It defines a set of general QoS characteristics, which are of importance to communications and processing systems. However, the QoS characteristics in the QoS framework are very general in nature and need refinement to be useful for specifying QoS constraints for a particular application domain. Table 5.3 shows an example how the general QoS characteristics (first column) are refined for audio/video QoS characteristics (second column), which can be used in the specification of a stream interface environment contract.

| QoS framework | Environment contract parameters | Example |
|---|---|---|
| Time delay characteristic (minute, sec, msec) | Jitter (variation in frames, samples)<br>Delay (frames, samples) | Jitter (± 1 frame, ± 100 samples)<br>Delay (3 frames late/in advance, 500 samples late/in advance) |
| Throughput characteristic | Application information throughput (fps, samples/s) | Throughput (25 fps, 8000 samples/s) |
| Service commitment | Guarantee level | Best effort service |

Table 5.3: QoS characteristics related to the ODP environment contract for multimedia QoS

An important aspect of QoS characteristics described in the environment contract is a description of the *guarantee level* provided by a computational object at its interfaces for each QoS parameter-value tuple. The object (and its interfaces) should obey a guarantee level that has been agreed upon. If the object can not maintain the agreed service commitment, the objects involved in using this service should be promptly informed.

The ISO-QoS framework [81] and Ferrari's Tenet project [105] distinguish three classes of service commitment. These are:

- *Compulsory (or deterministic) service*: A service is provided with a deterministic guarantee. This means that in absence of total failures (e.g. electricity cut-off, breakdown of a switch), the QoS requirements offered to a client of the service are met, such as maximum response time, availability etc. Fixed resource reservation must take place to be able to provide such a service;
- *Statistical reliable service (or Guaranteed service*): A service with a certain percentage of QoS violations is provided. Resource reservation is required, but now these resources can be shared with other users of this service;

- *Best effort service*: The request from a client for a certain service is evaluated against the current usage of the server object. If conditions change in future, the QoS provided to this client may change as a consequence.

A guarantee level can be assigned to each individual parameters in an environment contract, or a certain guarantee level can be reserved for the interface or object as a whole. The approach followed in this thesis allows for a flexible (parameterised) service commitment specification for each QoS parameter. This flexibility is desirable if, for example, an object supports a stream interface: a best effort guarantee level for the throughput but a statistical reliable guarantee for the reliability parameter.

### TINA-ODL specification of stream interface environment contract

The previous paragraphs described the QoS characteristics, which are desirable to include in an environment contract for multimedia applications. These characteristics need to be expressed in a (formal) implementation independent form, which can then be compiled to a specific target environment (e.g. ORBIX, a CORBA implementation from IONA). The TINA-ODL is used for this purpose and the syntax description for the environment contract specification is depicted in Table 5.4

| | |
|---|---|
| <environment_attribute> | ::= <environment_attr_type> <environment_attr_name> |
| <environment_attr_type> | ::= <simple_type_spec> |
| <environment_attr_name> | ::= <simple_declarator> |

Table 5.4: BNF syntax specification for environment contract in TINA-ODL

The stream interface environment contract determines the values for the QoS parameters as defined in the type definition for each flow. For each subtype, it describes the possible interval depending on the guarantee level requested. (i.e. deterministic, statistical or best effort). The following example shows an environment contract declaration. `VideoQos` is declared describing the throughput, jitter, errorrate and delay characteristics. Table 5.5 shows that depending on the guarantee level different values will be provided.

```
/* Environment contract description for a stream interface */

typedef enum { Deterministic, Statistical, BestEffort } Guarantee;

struct VideoQoS {
union Throughput switch (Guarantee) { /* expressed in frames per seconds */
   case Statistical:        float      Mean;
   case Deterministic:      float      Peak;
   case BestEffort:         struct Interval {float min; float max; }; };

union Jitter switch (Guarantee) { /* A time variation a frame gets through a service */
   case Statistical:        float      Mean;
   case Deterministic:      float      Peak;
   case BestEffort:         struct     Interval {float min; float max; }; };

union Erorrate switch (Guarantee) { /* in frames that are lost */
   case Statistical:        float      Mean;
   case Deterministic:      float      Min;
   case BestEffort:         struct     Interval {float min; float max; }; };

union Delay switch (Guarantee) { /* delay */
   case Statistical:        float      Mean;
   case Deterministic:      float      Maximum;
   case BestEffort:         struct     Interval {float min; float maxd; }; };

float priority /* 1: high priority - 1+x: low priority */
};

struct AudioQoS { /* similar to that for video */
union AudioVideoSynchrSkew switch (Guarantee) { /* Expresses skew between audio and video in ms */
   case Statistical:        float Mean;  /* the average skew */
   case Deterministic:      float Peak;  /* the maximum skew that is allowed */
   };
```

Table 5.5: TINA-ODL specification of stream interface environment contract

In this example, the QoS specification corresponds either to a *required QoS* clause as it would be for a video sink flow, or to an *offered QoS* as it would be the case for a source flow. Table 5.5 also shows that relations between flows can be expressed in an interface environment contract. For example, the `AudioVideoSynchskew` describes the skew interval between two flows in a stream interface, which have to be synchronised (inter-stream synchronisation).

Use of TINA-ODL syntax for the environment contract is rather flexible. It allows various ways to express the QoS constraints on interfaces and objects. This flexibility is necessary since the environment contract has to accommodate many different types of objects, which operate in different circumstances and thus require different QoS specifications. The QoS statements made in the computational specification should be fulfilled by the underlying engineering mechanisms. Thus the computational expression for a certain jitter boundary for video frames with a deterministic guarantee requires special engineering solutions to meet that requirement. See also Section 5.3, which discusses the correspondence between computational specifications and engineering configurations.

### 5.2.2 Typing and subtyping of stream interfaces

To determine whether two stream interfaces can be bound or can replace each other, there is a need for typing and subtyping rules. Typing rules for stream interfaces enable to check whether stream interfaces can be bound, and the subtyping rules enable to check whether another can replace a stream interface. The typing and subtyping rules described here are at the computational level. This means that they describe rules for typing stream interface templates prior to the actual binding of stream interfaces. This prevents most mismatches or runtime binding errors. This section details and illustrates these rules.

#### Subtyping of stream interfaces

Stream interfaces can be registered with the trader (like operation interfaces) and thereby be made available to other objects that might wish to communicate with the stream interface. In the case of trading interfaces but also for the actual binding of interfaces there is a need for defining compatibility between interfaces. For instance, if stream interface A can be a substitute for stream interface B, and a request for stream interface B can not be met explicitly, the trader can propose stream interface A which includes everything that stream interface B offers and maybe more.

In ODP-RM, the following subtyping rules are defined for stream interfaces for a primitive binding:
Stream interface A' is a subtype of stream interface A if for each flow having identical name the following holds:

1. If the causality is producer, the flow type in A' is a subtype of the flow type in A;
2. If the causality is consumer, the flow type in A is a subtype of the flow type in A'.

The compound binding object is responsible for all conversions between flows, and even for dangling flows (i.e. flows that are not bound) in a compound binding. This is different from a primitive binding where all flows have to obey the subtyping rules described above. Due to the different nature of a compound binding, its subtyping rules are slightly different. More precisely, the statement 'having identical name', can be relaxed in a compound binding because two flows with different names could be of the same type and therefore having a subtype relation. Two flows with different names but which obey the other two clauses can have a subtype relationship. The flow name is less important for subtyping if compared to operation names in operation signatures. In operation interfaces each operation in the signature is unique. For stream interfaces the ability to consume and produce a certain flow type with an associated QoS is of importance and can be expressed separately from the flow names.

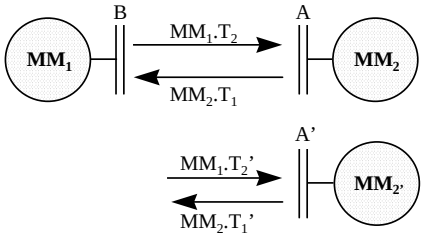



Figure 5.4: Subtyping of stream interfaces

The rules stated above are illustrated in Figure 5.4. A' (stream interface of object $MM_{2'}$) is a subtype of A (stream interface of object $MM_2$) if and only if it can replace A. Thus B (stream interface of object $MM_1$) is not aware of it. This means that the producer flows in signature A' should be equal or comprise the producer flows in signature A. Thus signature A' should not transmit flows ($MM_2.T_1$') of a type that is not supported by signature A ($MM_2.T_1$). The other way around, the consumer flows supported by signature A' ($MM_1.T_2$') should be at least the ones supported by signature A ($MM_1.T_2$ ) but may support more consumer flows than signature A. Figure 5.5 gives an example of A and A' flow types.

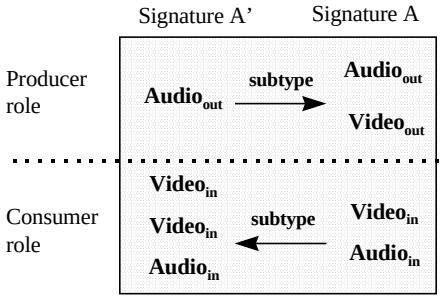



Figure 5.5: Example of subtyping for stream interfaces

## Typing of Stream interfaces

The typing rules are used during the binding process as described in Section 5.2.3. They enable a pre-check of stream interface template matching.

Stream interfaces are typed and each flow, which is part of the stream interface, has a type that can be supported by the available underlying mechanisms. Examples of basic flow types are individual audio and video flows. However, flows can be more complex and can represent arbitrary complex types.

Besides flow type compatibility, the attributes-value pairs in the environment contract should be compatible (i.e. QoS offered/QoS required description), and the behaviour of

the two stream interfaces should be compatible as well. This is a crucial requirement for successful binding of stream interfaces. It means that two stream interfaces can bind if

1. They have a compatible behaviour and compatible environment constraints;
2. There exist two flows to bind having compatible information type, compatible flow QoS and an opposite causality;
3. Two flows have compatible QoS, if the QoS required by the consumer flow is less strict than or identical to the QoS offered by the producer flow.

Figure 5.6 illustrates a possible binding of two multimedia objects using a binding object that will multiplex the $audio_{out}$ and $video_{out}$ flows of multimedia object A onto a $TV_{in}$ flow for multimedia object B, and vice versa the $TV_{out}$ is demultiplexed into an $Audio_{in}$ and $Video_{in}$ flow.

They obey the typing rules thus the following holds:

- 'TVFlowtype' has a subtyping relationship with 'AudioFlowtype' and 'Videoflowtype' (see Table 5.2);
- Signature B is a subtype of signature A and the flows that are bound have opposite causality;
- QoS: TVQoS requiredQoS is less strict than or identical to (Audio offeredQoS and Video offeredQoS);
- Audio requiredQoS and Video requiredQoS is less strict than or identical to TVQoS offeredQoS;
- Behaviour: CBR for both audio and video (8000 samples/s, 25 frames/s). Similar CBR for TV flow.

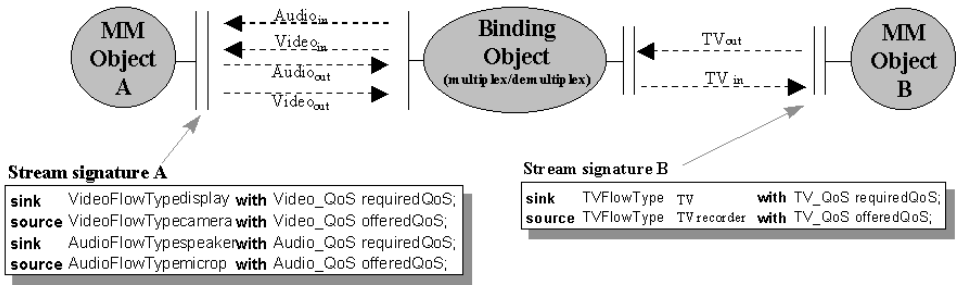



Figure 5.6: Example of binding stream interfaces

### 5.2.3 Multimedia Binding Object

Explicit binding and binding objects provide the basis for building complex communication configurations that interconnect stream interfaces. Examples of communication configurations are point-to-point communication, multicast communication, group communication, conferencing and bulletin-board

communication. This section focuses on a particular class of binding objects that are useful for distributed environments where the objects exchange time-based media.

This section first identifies the requirements for multimedia binding objects that should be satisfied in the specification of binding objects for multimedia processing environments. Then it describes the binding process and identifies generic control interfaces for multimedia binding objects inspired by the consortia described in Chapter 4. An example is shown how a compound multimedia binding object can be decomposed into smaller objects.

### Requirements on the multimedia binding object

A multimedia binding object is a specific binding object. It abstracts from the communication aspects between computational objects and the underlying technology to support multimedia exchange. Implicit binding (see Chapter 3) is not sufficient when interactions between computational objects become more complicated and impose additional requirements. Implicit binding does not require involvement of the client and server objects and they have no control over the binding that is established. This might be satisfactory for a large group of applications but it is insufficient for multimedia applications and co-operative applications [106]. Multimedia applications require that resources are allocated in the nodes and also in the network to deliver QoS guarantees as specified in the environment contract of the multimedia application. This means that multimedia applications require explicit control over the (multiparty) binding that is to be established, to meet time-critical delivery of the time-based flows and operations.

In the case of multimedia interaction, the following requirements can be put on the binding between computational objects. These requirements are primarily from the user or application designer's perspective.

- The binding object should be able to bind two or more stream interfaces that are of a different type. It should be possible to bind a stream interface supporting audio and video flows to another stream interface supporting only audio. The binding object can allow flow manipulation. For instance, flows (of possibly different types) from a number of producers can be combined to provide a composite flow to a single consumer;
- The binding should support negotiation of QoS between the computational objects involved. Multimedia requires specific QoS (described in the object specification, see Table 5.5). The binding object should support and, if necessary, be able to (re)negotiate the QoS for the computational objects involved;
- The computational objects should be able to control and monitor the binding that is established. The binding should also support dynamic adding and removing of interfaces that are involved in the binding;
- It should be possible to express the configuration of flows that have to be bound. It is desirable to express which outgoing flow has to be bound to which incoming flow;

- The binding should support group communication. This implies that a flow (or operation) sent by one of the bounded interfaces is simultaneously delivered to a group of receiving interfaces. Group communication is especially useful for multimedia conferencing services where participants should receive the audio and video flows or operations simultaneously;
- The multimedia service can explicitly request the binding object for the allocation of resources in the transport network to meet requested QoS guarantees.

Bindings have to be implemented and therefore the DPE programmer imposes additional requirements (or constraints): The binding object for stream interfaces should be as simple and generic as possible because it has direct influence on the supporting engineering infrastructure. If certain properties are imposed on the binding object it might be impossible to realise the binding object (e.g. require an audio to video flow conversion). It is desirable that the engineering model of computational stream binding objects remains similar in complexity compared to that of a computational operational binding. This facilitates the automatic translation of the computational binding object onto an engineering model.

**Binding process**

The general process for explicit binding is already explained in Chapter 3. This is used as a basis to describe a scenario for the explicit binding of three computational objects similar to the configuration shown in Figure 5.7. In this example, the server object (TV-object) interacts with the binding factory object. It invokes a `bind()` operation to request for the creation of a binding object that corresponds to a specific binding object template (e.g. three-party TV/Radio/AV template). A parameter in the `bind()` operation is a list of stream interface references of the stream interfaces to be bound. The stream interface reference contains all the information needed by the binding factory to determine whether the interfaces can be bound according to the typing rules for stream interfaces (see also Section 5.3.4). The initiating object can specify a *logical connection graph* in the `bind()` operation that describes the topology of how the flows in the stream-interfaces have to be bound (see also Section 4.3).

The binding factory creates the multimedia binding object to enable data exchange between the involved objects. The multimedia binding object instantiates several control interfaces through which it can be controlled by the server object (Figure 5.7, ③), the client objects (④) and possibly other computational objects (⑤) not directly involved in the flow exchange. The control interface (⑤) is used, for example, by the QoS manager to monitor the QoS and to initiate operations (e.g. QoS-renegotiation) in case of QoS contract violations. The binding object can also invoke signal operations (not shown in Figure 5.7) on the clients and servers to notify particular events (e.g. synchronisation events). The binding object should realise the QoS guarantees between the interacting interfaces as described in the QoS environment contract. The references of the control interfaces are returned to the objects involved in the binding.

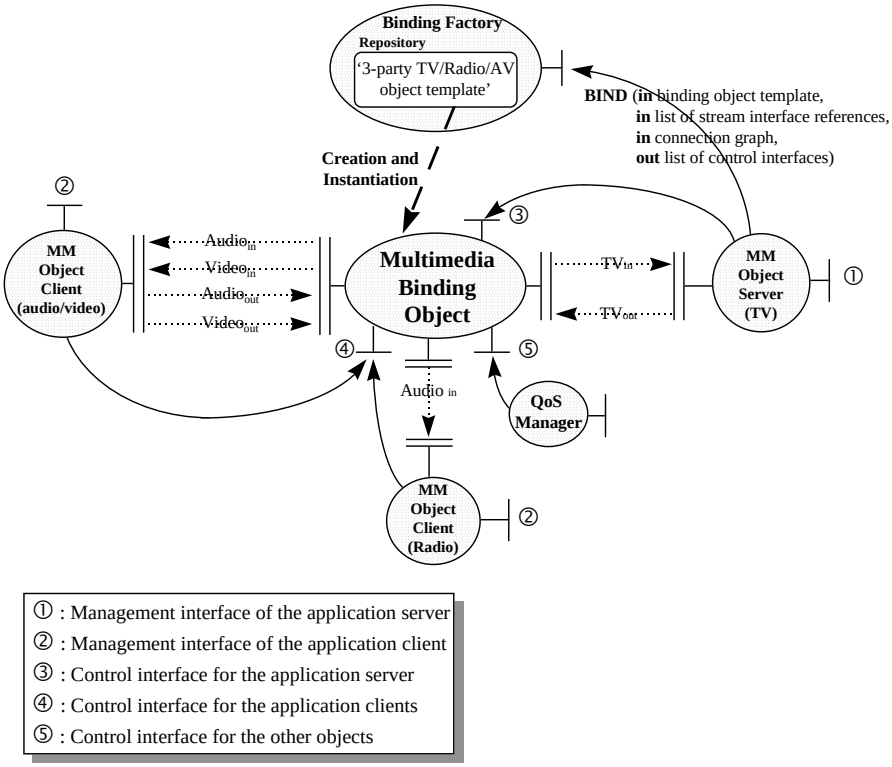



Figure 5.7: Binding process for multimedia binding object

After instantiation of the multimedia binding object, producer and consumer flows are linked and controlled by the multimedia binding object. The functional behaviour exhibited by the multimedia binding object is specified in the 'behaviour' part of the binding object template and in this example consists of multiplexing audio and video flows onto a composite TV flow and vice versa demultiplexing TV flows into separate audio and video flows.

## Generic control interfaces

As seen in Figure 5.7, a multimedia binding object offers several management and control interfaces, which are detailed in Figure 5.8. The consortia described in Chapter 4 identify solutions for multimedia services, and several of them define control interfaces, for example TINA and IMA. To achieve hybrid standardisation, this section shows how several of these de-facto specifications can be used to specify the control interfaces of a multimedia binding object.

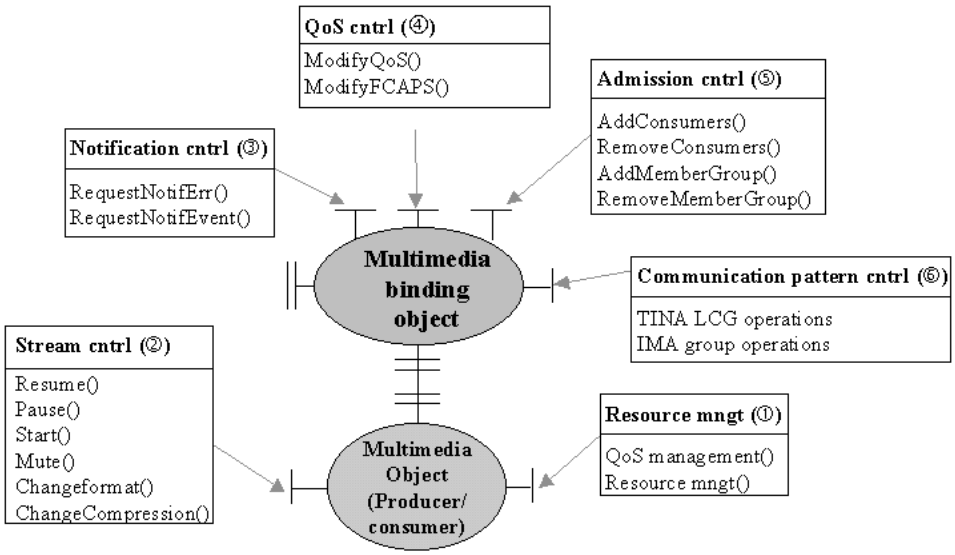



Figure 5.8: Control and management interfaces related to multimedia (binding) objects

*Resource Management Control interface* (Figure 5.8, ①): is provided by multimedia objects to perform resource and QoS management operations on the multimedia devices. This interface is used by clients of multimedia devices to reserve physical resources required for the multimedia device(s). Resources include both system resources, such as CPU, memory, as well as specialised multimedia resources such as audio and video devices. The client wants to get guarantees with respect to the performance of the multimedia device(s). The client must also specify the desired QoS when requesting a multimedia device. In IMA specific QoS characteristics are defined that can be used by a client to specify the QoS [33]. For example, the guarantee level, the reliability of data delivery, delay bounds, jitter bounds, and bandwidth bounds.

*Stream control interface* (②): is used to perform operations on the multimedia device(s) to control the stream of data. This interface contains operations to control the stream progress and to determine the stream position. Operations include pause, mute, stop and start operations. The stream control operations should be performed on single flows in the stream interface. Therefore the stream control operations contain an input parameter specifying the flow on which the operation should be performed. IMA specifications give detailed IDL specifications of stream control operations (i.e. IMA `stream` interface [32]).

*Notification control interface* (③): generates notifications of events of interest to the application, which are detected by the engineering objects supporting the binding. For example, an event might be signalled at the start or end of a period of silence in an audio flow. The OMG event notification service [147] can be used by the binding object to notify interesting occurrences such as QoS violations to its clients.

*QoS control interface* (④): provides operations for the dynamic control of the QoS associated with the binding. Operations are defined to manipulate QoS parameters (e.g. jitter, bandwidth, throughput). IMA specifics interfaces related to the control of time-based flows: the `SyncStream` interface to keep multiple flows synchronised, the `controlled` stream interface to control the advance of a flow, and the `Group` interface to control the end-to-end QoS and resource acquisition.

*Admission control interface* (⑤): supports operations needed for the provisioning of a dynamic multicast binding, in which information is distributed to a set of consumers which can vary in time. Operations are defined, which add additional consumers or remove existing consumers from the binding. IMA specifics specific operations for the manipulation of the binding (i.e. `Group` interface [32]). The admission control interface is closely related to the QoS & Resource control interface of the binding since manipulation of the binding might have an impact on the resources and the end-to-end QoS provided by the binding.

*Communication pattern control interface* (⑥): is used to manipulate the existing configuration of flows that are bound by the binding object. It provides operations to modify, delete connectivity relations. TINA specifies a specific interface called the `LogicalConnectionGraph` interface (see Section 4.3.2), which is specific for telecommunications applications. This solution is based on graphs that describe the relations between the stream interfaces involved. IMA has a rather similar approach and also defines graphs, which describe the relations between virtual resources part of a virtual connection (called a group). IMA defines operations to add and remove virtual devices from the graph. Naturally, the details of both the TINA and IMA operations regarding the graph manipulations are different.

Discussion: The IMA specifications provide detailed specifications of control interfaces related to their specific multimedia binding object (i.e. IMA `virtual connection` object), which is quite different from the ODP multimedia binding object (see Section 4.2). However, these control operations are based on the model promoted by IMA and not directly applicable to all kinds of multimedia bindings (e.g. multipoint-to-multipoint bindings is not supported by IMA). In general, this is the case for most binding objects that will be developed by consortia that have a specific operating environment in mind. For example, the TINA LCG interface is not applicable to multimedia binding objects outside the telecommunications domain due to the use of the CSM object (see Section 4.3). As a consequence, the operations of the control interfaces described above need to be detailed for each particular multimedia binding object, which depends on the environment where this object is used.

### Example of refinement of the multimedia binding object

The multimedia binding object enables the modelling of complex multipoint-to-multipoint configurations, binding stream interfaces with different QoS and/or flowtype

descriptions. Such a complex binding will be decomposed into simpler entities necessary to deal with multiparty flows, such as basic bindings and (if available) special resources such as a videobridge. For the decomposition of a compound binding different solutions are possible depending on the objects to be bound and the availability of special resources. This implies that control interfaces of a compound binding will not only be service specific but also specific to the particular service instance while other control interfaces are generic.

The instantiation of the compound binding object corresponds to a sequence of actions to be performed by the object factory. The actions depend on the decomposition of the compound binding template. Figure 5.9 shows a possible decomposition of the compound multimedia binding object into a video bridge object and a videobridge controller.

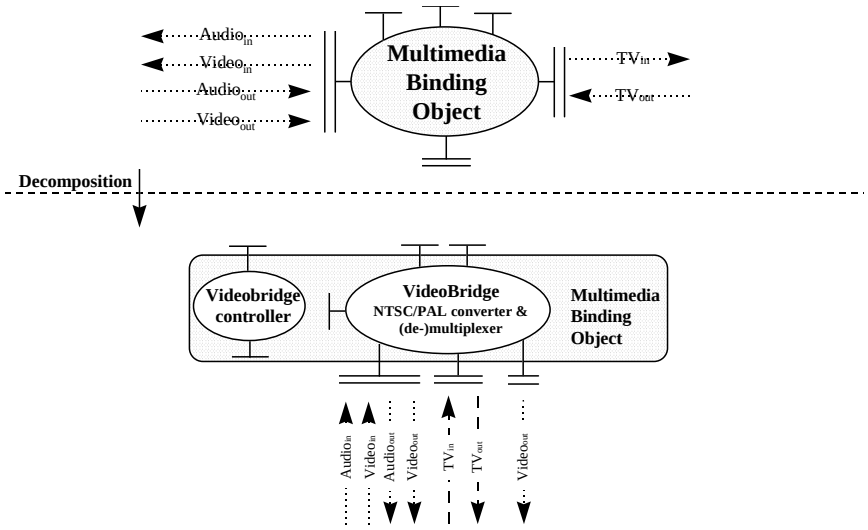



Figure 5.9: Objects involved in a multiparty audio/video exchange.

The videobridge is in charge of converting and multiplexing the flows. Control interfaces that are part of the multimedia binding object can be dispatched over several computational objects. Figure 5.9 only shows a videobridge controller object but additional objects can be present as well.

## 5.3 Engineering support for multimedia services

The requirements expressed in the computational viewpoint with respect to stream interfaces, explicit binding and QoS require a supporting infrastructure to fulfil these requirements. Figure 5.10 shows an engineering configuration of objects and highlights the different correspondence between computational and engineering concepts to reflect the computational multimedia characteristics. The ODP engineering viewpoint focuses on the infrastructure and mechanisms to support distributed processing.

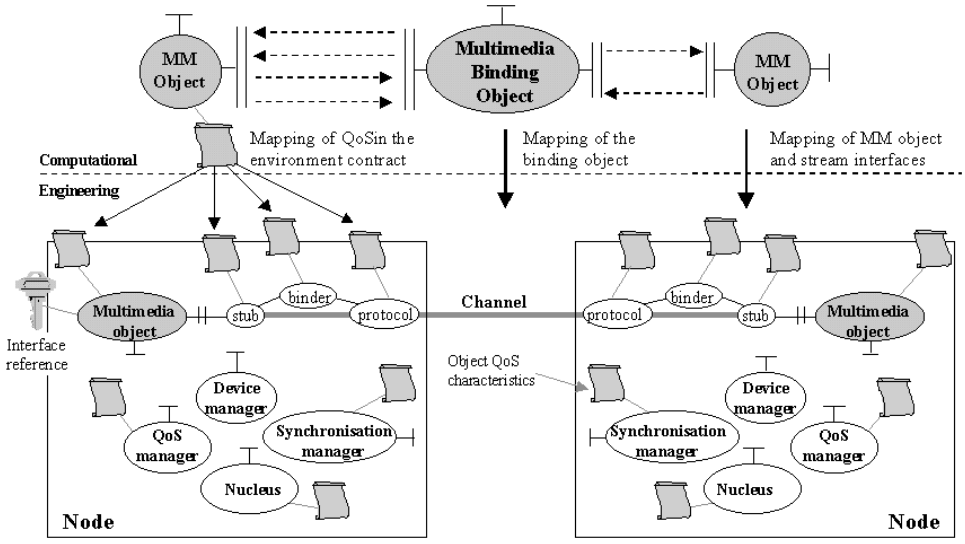



Figure 5.10: Engineering configuration and relations with the computational specification

The translation of computational objects with operation interfaces onto an engineering configuration is supported by most DPEs (described briefly in Chapter 3). However, use of the same translation for stream interfaces is not appropriate due to the different nature of stream interfaces [51]. The author proposes in Section 5.3.1 an engineering configuration that is more suitable to exchange time-based media.

The computational environment contract expresses the QoS required and offered which have to be supported by the engineering infrastructure. In this context, an open issue is how computational QoS specifications relate to engineering objects and associated QoS characteristics. Section 5.3.2 focuses on QoS properties related to multimedia interaction and gives an overview of possible relations between computational and engineering QoS specifications.

Figure 5.10 shows several supporting engineering objects that are relevant for nodes dealing with multimedia data. It is expected that several of these engineering objects

will be present in nodes supporting multimedia. Section 5.3.3 identifies several generic QoS characteristics and properties in the context of de-facto and de-jure standards, which are desirable for these objects. It is beyond the scope of this thesis to discuss all the details of these objects.

Information about the characteristics of stream interfaces needs to be exchanged between nodes that are involved in the binding. An open issue is which information needs to be exchanged for stream interfaces (via the stream interface reference). In particular, QoS plays an important role for stream interfaces and it is essential to be able to exchange information about QoS to guarantee the computational environment contract. Section 5.3.4 proposes a general stream interface reference structure apt for DPEs.

The multimedia binding object abstracts from distribution aspects. In the engineering view, this should be solved by providing functions needed to support the binding object. As indicated previously, the engineering configuration of multimedia binding objects is partially designers' concern and is often implementation and platform dependent. Nevertheless, several translation rules are described in Section 5.3.5 that can be used for the mapping of multimedia bindings.

Section 5.3.5 relates the computational binding process to the engineering channel establishment. It discusses the channel establishment for streams that apply to a broad set of DPEs that need to incorporate time-based media.

### 5.3.1 Engineering representation of computational objects supporting stream interfaces

A computational object supporting a stream interface can be regarded as a composite object that encapsulates two or more objects (Figure 5.11, computational part). It generally encapsulates a virtual device able to process the time-based flows, and a computational object that manages this device. Figure 5.11 shows a possible decomposition of the multimedia object into an audio-visual device object and a management object. These objects can be further decomposed. The computational management object corresponds to an engineering BEO (Figure 5.11, Mngt-object). The computational audio-visual device corresponds to an engineering display device object and speaker device object. Additional engineering objects are also visible in the engineering configuration such as a device manager, and the channel objects (i.e. binder, stub and protocol object)

Figure 5.11 shows an engineering configuration for computational objects that support stream interfaces, which is different from those that support operation interfaces (see Figure 2.10). The reason is that for operation interfaces the channel receives invocations from an object at the client side, and sends the invocations to the server side where it is delivered to another object. The operation channel configuration is built on the assumption that the objects process information, where the action/reaction chain of

invocations and processing form together the distributed application. This assumption does not hold for the exchange of time-based flows (see also discussion in Section 5.2.1).

In many multimedia systems the application objects do not process the time-based flows themselves but only control the flows by requesting and relating a set of devices that are in the domain of a device manager. A device is an object that is involved in producing or consuming time-based flows. These devices can be typical available on a node. In general, the set of devices falls into two classes: devices concerned with transport of information over networks, and devices concerned with processing of information. The protocol object is an example of a transport device. Speakers, microphones and video cameras are examples of processing devices.

The specific configuration of related devices make up a distributed multimedia application. Figure 5.11 (engineering part) shows this principle where the different objects are located in different capsules to allow other application objects to share the audio-visual devices.

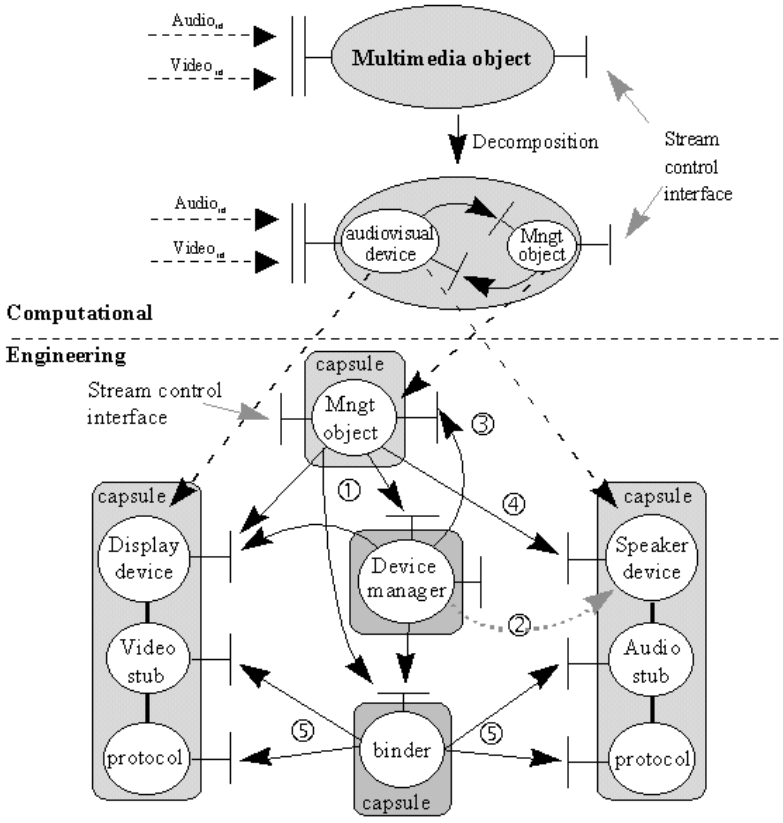



Figure 5.11: Correspondence between computational objects and engineering objects

For the exchange of time-based flows, the stream channel consists of one binder object and a pair of stub/protocol objects for each time-based flow. Also, the device objects and the management object for stream channels are positioned in different capsules. This in contrast with operation channels, where a chain of management, stub, binder and protocol objects is positioned in the same capsule. For stream channels, the binder object manages the channel halves but does not process the flows. This is the task of the stub and protocol objects that are selected from a library in such a way that their characteristics match the computational stream interface definition (e.g. an MPEG flow needs an MPEG codec functionality offered by a stub object).

A scenario for the control of time-based flows in the engineering viewpoint can be as follows. The management object claims a device via the device manager (Figure 5.11, ①), for instance a speaker device. The device manager instantiates the speaker device object (②), and passes a control interface reference of the speaker device to the management object (③). The management object can now directly control the speaker device (④). In the case of stream interfaces the multimedia object only manages the processing devices and delegates the control of the protocol and stub objects to a binder object (⑤). The binder object manages these objects that are active in a stream binding and reacts on reconfiguration requests, e.g. in the case of changing QoS.

### 5.3.2 Engineering correspondence of computational environment contracts

A computational QoS specification in an environment contract is expressed in terms of media characteristics and media relations, which do not make any reference to engineering or technology mechanisms. A translation should be made onto the engineering configuration, which realises the QoS properties specified in the computational environment contracts. A computational QoS specification covers both the end-to-end aspects and the QoS aspects of the end system's node (as reflected in the offeredQoS and requiredQoS description of the computational environment contract).

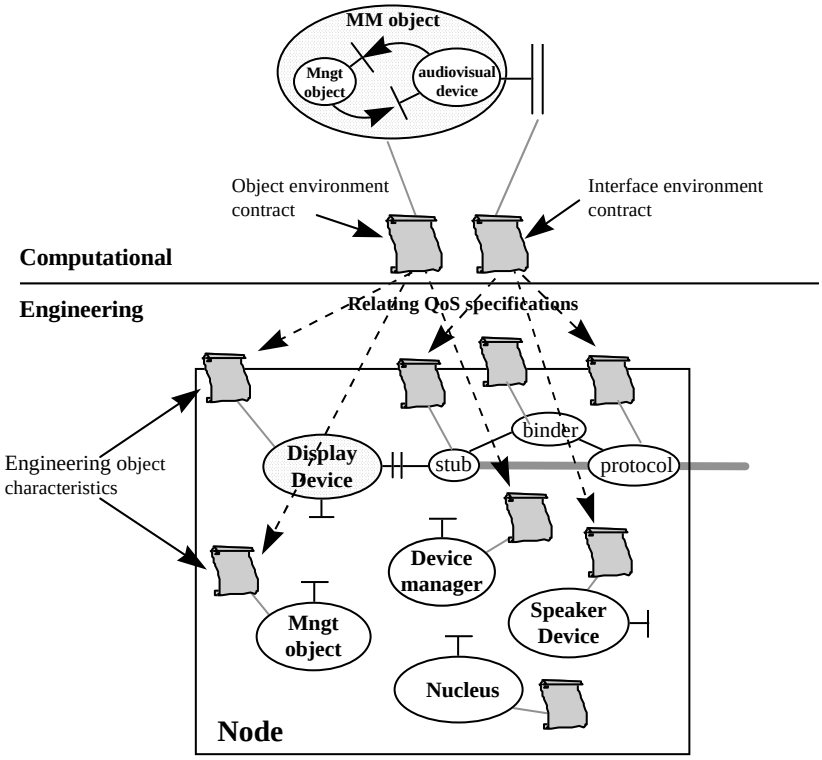



Figure 5.12: Relating computational environment contract to engineering QoS characteristics

Figure 5.12 shows the relation between the computational and engineering viewpoint descriptions related to QoS specifications. An important step in the translation process is to identify the engineering objects, which correspond to the computational multimedia object, as well as, the characteristics of the engineering objects. The term *'engineering object characteristics'* is introduced to describe the QoS characteristics of the engineering objects. These characteristics are mainly determined by the technology that is used to implement the engineering object. For instance, a stub object providing MPEG coding/decoding functionality has specific fixed characteristics (e.g. coding/decoding time needed for a single video sample). The values of the object characteristics may vary during the lifetime of the engineering object and are influenced by other engineering objects that run on the node, or by QoS variations within the network. For example, de decoding time for an MPEG video frame slows down if less CPU time is available for the stub object.

The engineering object characteristics can be classified into two groups. First, characteristics related to computing systems and expressed in a *system QoS* specification. It describes typical system QoS characteristics such as, buffersize, memory and operations/s. Second, characteristics related to the network and expressed in a *network QoS* specification. It describes typical network QoS parameters such as the throughput, jitter and delay characteristics of the network used.

These object characteristics are used in the QoS translation process, as specified in the computational environment contracts. This translation is not trivial, since in general it is not a one-to-one translation. A single computational QoS parameter can be translated into multiple engineering characteristics. To determine these translations, techniques such as queuing theory or operations research [13][103][104] could be applied.

It is outside the scope of this thesis to formally determine the translations between computational QoS parameters and engineering characteristics. An example of how a computational QoS parameter translates onto engineering characteristics is the following: The computational specification of throughput, expressed by the number of frames/sec will be translated to throughput in Mbit/s for the protocol object and compression/decompression speed for the stub object in the engineering viewpoint. Furthermore, the display device needs per invocation (for the processing of a video frame) a number of milliseconds on a certain CPU type located in the nucleus.

The engineering object characteristics are used in the allocation process to check whether a suitable amount of resources from the nucleus is allocated to the capsules. The values of the QoS characteristics specify the available capacity of the nucleus and the engineering objects. The QoS characteristics associated to the channel objects guide the allocation of communication resources in order to realise the required throughput, delay and jitter [103].

As stated in ODP-RM, the viewpoint specifications are not independent and there should be a correspondence and consistency between the viewpoint specifications. Doing this for the computational environment contract is outside the scope of this thesis and stays an open issue. Ideally, a compiler exists, which takes into account the computational specification with a QoS description and automatically creates corresponding engineering configuration, which complies to the computational QoS specification. This is partially true for IDL compilers that compile a computational specification (expressed in IDL) into engineering pieces of executable code. However, the author does not know any compiler that also includes the QoS specifications. This implies that the engineering specification needs to be checked manually, and probably be extended with additional objects (e.g. QoS managers) and mechanisms (e.g. QoS negotiation protocols, QoS monitors) to guarantee the computational QoS description.

### 5.3.3 Engineering objects

This section presents engineering objects involved in multimedia services. The objects discussed are the nucleus, stream channel objects and additional supporting objects such as multimedia devices and the synchronisation object. It is beyond the scope of this thesis to specify the objects in detail except for the synchronisation object that is described in Chapter 6.

### Nucleus

The nucleus is responsible for the processing, storage and communication resources within a node. The nucleus is related to the operating system of the node. It shields the computer hardware from the software that is running on a node. The nucleus offers various services for other resources in a node such as CPU, memory, storage and input/output devices. To deal with multimedia adequately, additional services from the nucleus are required for the processing of time-based flows.

Traditional operating systems are designed for non-time critical applications. Operating systems that support time-critical functions need to offer performance guarantees and must have the capability to exercise control over the allocation of system resources such as the CPU. This necessity takes a *process manager* to deal with the allocation of processor resources. The process manager maps a single process onto the processor resource according to a specified scheduling policy, which takes into account requests imposed by other processes. An important aspect of real-time multimedia systems is the need for timeliness. For time-based flows, it is important that the data is processed regularly and the correct presentation of the flow fails when the system is not able to process the workload in time.

To fulfil these timing requirements for time-based flows, the nucleus must use real-time scheduling techniques. Requirements for scheduling time sensitive data are different from non-real-time processes. For many applications dealing with time-based flows, missing a piece of information is not always a severe failure. For example, if a video sample can not be decompressed in time it is better to drop it as soon as this does not happen for a long sequence of frames. A sequence of digital time-based media results from periodically sampling a sound or image signal. Therefore, in processing the data units of time-based flows periodic scheduling is needed [89].

The *Node management* function manages the threads in a node. Several threads will be active to deal with time-based flows on a node. In general, for each flow a separate thread will be running since the devices are located in different capsules [51]. Due to the timeliness of these flows, threads have to be scheduled so as to satisfy the timeliness constraint. Customised thread scheduling algorithms can be used for real-time exchange. Pre-emptive scheduling is also appropriate for real-time flows, where a running thread can be stopped for another thread to resume. Various pre-emptive policies can be applied such as First In First Out, Round Robin or Time-sharing [56].

The *Resource manager* located in the nucleus plays an important role for the determination of the load in the node and to determine whether the QoS characteristics requested in the computational environment contract can be satisfied. It has a complete view of the availability of resources on a particular node (devices, CPU, memory, etc.).

The *QoS manager* checks if the node can and will fulfil the agreed computational environment contracts. It uses monitoring and control functions to perform its task. It issues requests to the resource manager for resources (e.g. buffers, CPU-time, audio-device etc.) that must be allocated to a particular application. The QoS values of objects

running in the DPE can vary over time. Changing system load on the node can have an effect on the throughput value provided by a protocol object. Therefore, the node needs to be monitored frequently to check the actual QoS of its resources and if needed it must take corrective actions to prevent violation of agreed QoS contracts for already established sessions.

### Synchronisation manager

Audio and video exchange is an important aspect of distributed multimedia services and the ability to control and co-ordinate multiple audio/video streams is identified as an important requirement for DPEs. Chapter 6 discusses synchronisation in detail.

### Device objects

Device objects are engineering objects that represent either hardware devices (e.g. video capture board, display cards) or software devices (e.g. printer driver). Device objects represent a resource either internal the node or external (e.g. a set-top box) but under control of the node. From the engineering viewpoint, multimedia devices should abstract from technology implementations. Multimedia devices should provide a set of common interfaces that can be used by clients without being concerned with the specific hardware/software realisation of the multimedia device. Section 5.2.3 provides several general control interfaces for devices. Multimedia devices use system resources provided by the nucleus and the client of a multimedia device should be able to perform resource management operations on the device. Multiple media devices can be mapped onto a single hardware device. However, the client wants to have guarantees about the performance of the device and it uses the resource management control interface for this purpose.

### Stream channel objects

Streams require different functionality from a stub object due to the different nature of time-based flows that are exchanged. Stub objects should provide the mechanisms to encode and decode time-based flows. Additionally, data available for the producer or consumer of the flow should be notified. For this purpose the stream stub object provides control operations to local resources (e.g. buffer) and is able to notify particular events related to the flow (e.g. QoS change, no buffer space available, data drop out). Information is maintained by the binder object with respect to the required QoS.

In distributed systems, the RPC mechanism is widely used to implement operation interfaces. However, for the exchange of time-based, specialised *protocols* without the RPC mechanism are necessary. RPC requires that each buffer of data to be transferred is treated as a separate action with no particular relationship between previous and future RPC calls. Time-based flows require relationships between calls and a stream protocol is applied that creates a virtual channel between two protocol objects for the

duration of the flow exchange. In this case, relations between the data units of the flow can be defined specifically.

### 5.3.4 Engineering stream interface reference

In distributed systems, information about object instances and in particular information about their QoS requirements is passed using interface references (see also Section 2.3.4.2). Currently, most distributed systems only support operation interfaces and no explicit space is reserved in the interface reference for the exchange of QoS information.

This section focuses on the stream interface reference and shows how the computational environment contract can be included in this stream interface reference. Section 5.3.6 discusses how stream interface references can be passed in an ODP compliant environment. The QoS mechanisms that use the information in the interface reference to perform certain QoS management functions are not discussed. Work in this area is reported in [21][93].

#### Stream interface reference

A stream interface reference has the same general characteristics as an operation interface reference. Its multimedia characteristics appear mainly in the computational interface type that it represents, including the QoS specification and channel configuration of stubs, binders, protocol objects and possible interceptors needed. Figure 5.13 illustrates the general structure of a stream interface reference.

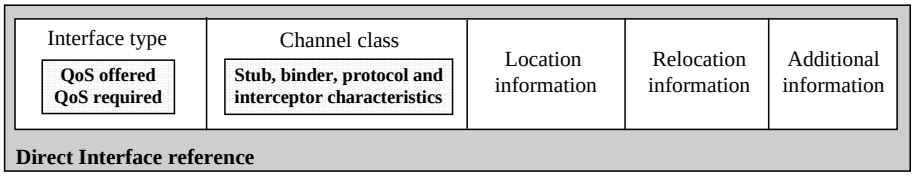



Figure 5.13: Generic Stream Interface reference

A refinement of the BNF definition for interface references specialised for stream interfaces using TINA-ODL syntax is shown in Table 5.6.

| | |
|---|---|
| <interf-type> | ::= <stream-interf-typ>\|<operation-interf-type>\|<signal-interf-type> |
| <stream-interf-type> | ::= <stream-interf-ref-name> {<flow-description> ';'}+ |
| <stream-interf-ref-name> | ::= <string_type> |
| <flow-description> | ::= <flow-name> <direction> <flow-type> <flow-QoS> |
| <flow-name> | ::= <string_type> |
| <direction> | ::= '**producer'**\|**'consumer'** |
| <flow-type> | ::= <type_spec> |
| <flow-QoS> | ::= <QoS_attr_type> <QoS_attr_name> |
| <QoS_attr_type> | ::= <simple_type_spec> |
| <QoS_attr_name> | ::= <simple_declarator> |

Table 5.6: Refinement of the BNF definition for stream interface reference

The `flow-description` contains information about each flow in the stream interface that reflects the computational stream interface specification. It describes the name of the flow (`flow-name`), the direction of the flow (`direction`), the type of flow (`flow-type`) and the associated QoS characteristics (`flow-QoS`).

The information in the interface reference is needed to determine whether two flows can be bound using the rules outlined in Section 5.2.2. Focusing on the flow-QoS, a possible C-structure is presented in Table 5.7 based on a compulsory QoS agreement, which implies that the values for the QoS parameters indicate the highest quality.

```
struct  { // computational oriented parameters
   long       sample-rate; // the sample rate for audio
   long       frame-rate;  // the frame rate for video
   ...
   // engineering oriented parameters
   long       throughput;  // bytes/s, max. throughput of the flow
   long       jitter;      // delay variation (msec)
   long       error-rate;  // magnitude order (10exp-)
   boolean    flow-control;// indicates if flow control is supported
   long       loss-rate;   // magnitude order (10exp-)
   long       delay;       // (msec)
   long       average-rate;// bandwidth (bits/sec)
   ...
   };
```

Table 5.7: Example C-structure for QoS description

### 5.3.5 Engineering correspondence of the computational multimedia binding object

The engineering configuration corresponding to the multimedia binding object as depicted in Figure 5.14 (upper part) can be of two extreme types (and all the intermediate ones). The first possibility is a centralised solution where one object receives and dispatches all flows (Figure 5.14, lower part). The second possibility is based on a completely decentralised configuration where several objects are responsible for receiving and dispatching the flows (Figure 5.15).

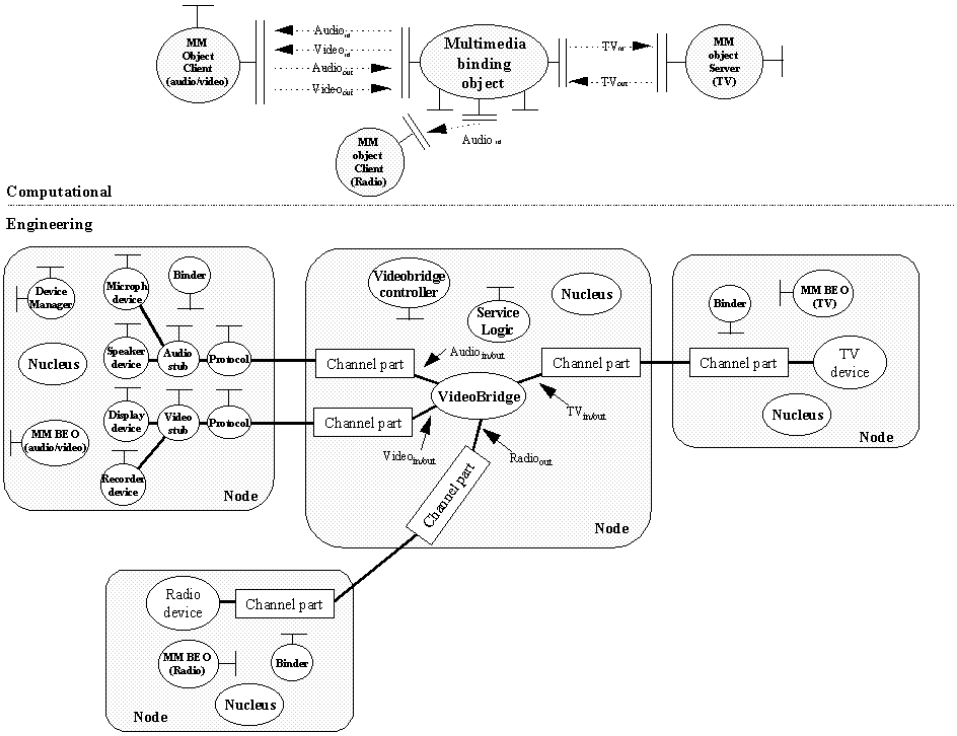



Figure 5.14: Centralised solution for the Binding object[13]

Figure 5.14 presents a centralised solution where all data passes through a central object (videobridge), which is located in a separate node with respect to the producing and consuming clients located in other nodes. The advantages of this configuration are:

- A central object such as a videobridge, which receives all flows and performs conversion between different flowtypes, is easier to achieve than performing this task in separate nodes involved in the binding. Also manipulation of flows such as multiplexing audio and video flows onto a TV flow is located in one place. These

13 The interactions on operation interfaces of engineering objects are not shown in this figure for readability purposes.

expensive manipulations are performed in one place, which avoids the need to have special equipment in the involved nodes;

- The multiparty binding is decomposed into several two-party stream bindings. Typing of stream interfaces is now reduced to the typing of pairs of stream interfaces;
- Using a central object provides the advantages of central control over the bindings that are established. This is useful in several situations. For instance, if the flows in the compound binding need to be synchronised and all the client object have to receive the flows at the same time a central object can help achieving this;
- The number of virtual connections grows linear when new multimedia objects are added to the binding. For each flow between the multimedia object and the central object, a connection has to be established. When another multimedia object is added, only connections need to be established between the central object and the multimedia object. Connections established previously between other multimedia objects are not involved.

The major disadvantage of the central approach is that when the central object fails the complete multimedia compound binding fails. If high reliability is demanded for the compound binding then additional measures have to be taken. The use of a central object implies that all functionality of manipulating flows is located in one object and this requires the availability of powerful hardware and software.

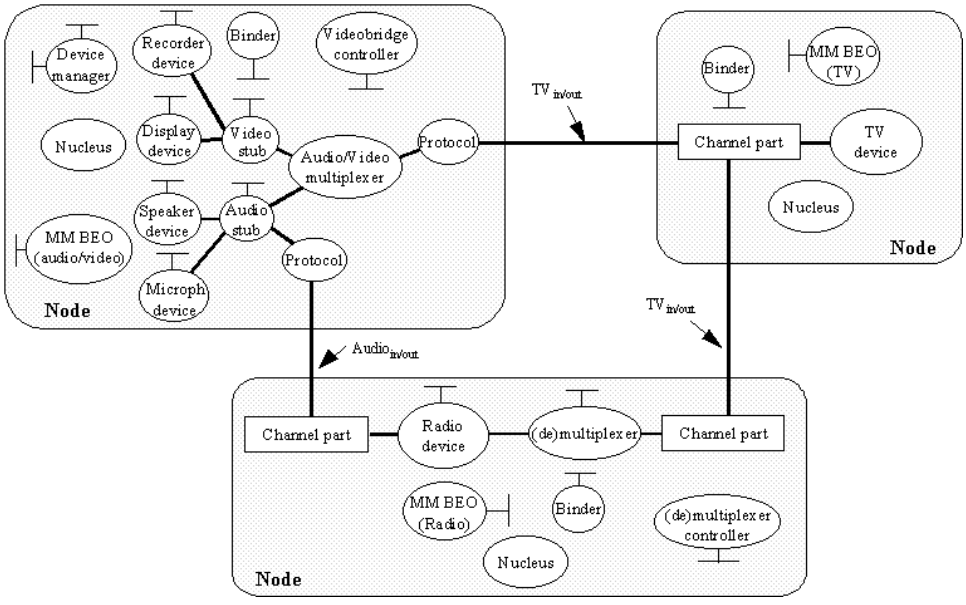



Figure 5.15: Decentralised solution for the multimedia binding object

Figure 5.15 depicts a decentralised configuration for the multimedia binding object. There is no central object that receives and dispatches the flows but instead each node that is involved receives and sends the flows. The central object in Figure 5.14 is now subdivided in several other objects such as audio/video multiplexer to create TV flows, and (de-)multiplexer located in the nodes. Each of the nodes receives the flows of the other nodes. Depending on the flowtype, manipulation of flows is performed in the separate nodes.

Advantages of this approach are:

- There is no central object that is in charge of all the flow manipulations. This requires less demanding hardware and software;
- A complex central object performing all the flow manipulations is not needed in a decentralised configuration. A translation onto hardware/software developed for the mass market is easier to achieve;
- The configuration is less sensitive to failures. If one of the nodes fails, communication is still possible between the other nodes;
- Reconfiguration of the binding is easier to achieve. If one of the nodes cannot maintain the required throughput for a certain connection then the flows can be redirected to another node(s).

A disadvantage of this approach is that all the nodes involved might need to have special objects to perform flow manipulations. If the nodes represent end-user systems it might be undesirable to force these nodes to have these special resources. However, the major disadvantage of this approach is the number of (virtual) connections that have to be set up to achieve a multiparty binding. If another (new) node is added it implies that several new connections have to be established between the nodes already present in the binding and the new node. The amount of virtual connections increases exponential and this will most likely cause problems in the existing nodes with respect to available resources.

### 5.3.6 Engineering correspondence of the computational binding process

The previous section described an engineering configuration of channels and engineering objects that correspond to the computational multimedia binding object. It did not discuss how the computational binding process is reflected in the engineering viewpoint by means of channel establishment. This section discusses this aspect and focuses in particular on the QoS that plays an important part for time-based flows. Channel establishment is described in the context of the ISO-QoS framework that identifies a set of QoS functions that perform activities such as QoS establishment.

The ISO-QoS framework describes several general QoS negotiation mechanisms that can be applied to distributed systems. The author focuses on one QoS negotiation mechanism called ‘Compulsory QoS’ and applies this mechanism to an ODP compliant

configuration as depicted in Figure 5.16. The stream interface reference is used as the means to exchange QoS information between the nodes.

The QoS framework defines three phases for QoS activity [54] that can be applied to ODP systems as follows:

- In the *prediction phase* (engineering) objects will make QoS enquiries to predict aspects of system behaviour so that the engineering objects can initiate QoS mechanisms appropriately. Enquiries could include, for example, the current loading of the node or previous levels of QoS achieved;
- In the *establishment phase* the engineering objects express requirements for QoS (as specified in the computational environment contract), enter into negotiations or re-negotiations, make agreements on the QoS to be delivered and offered. Furthermore, the actions to be performed if QoS degrades, and initiate mechanisms that will be needed during the operational phase;
- In the *operational phase* the agreed QoS will be monitored and maintained and if violated, appropriate actions will be taken.

The actions to be performed in the different QoS phases can be applied to the channel establishment between two or more stream interfaces residing on different nodes. Channel establishment is described in general terms in ODP-RM and it is refined here by the author, identifying the set of operations specific for the channel establishment for stream interfaces.

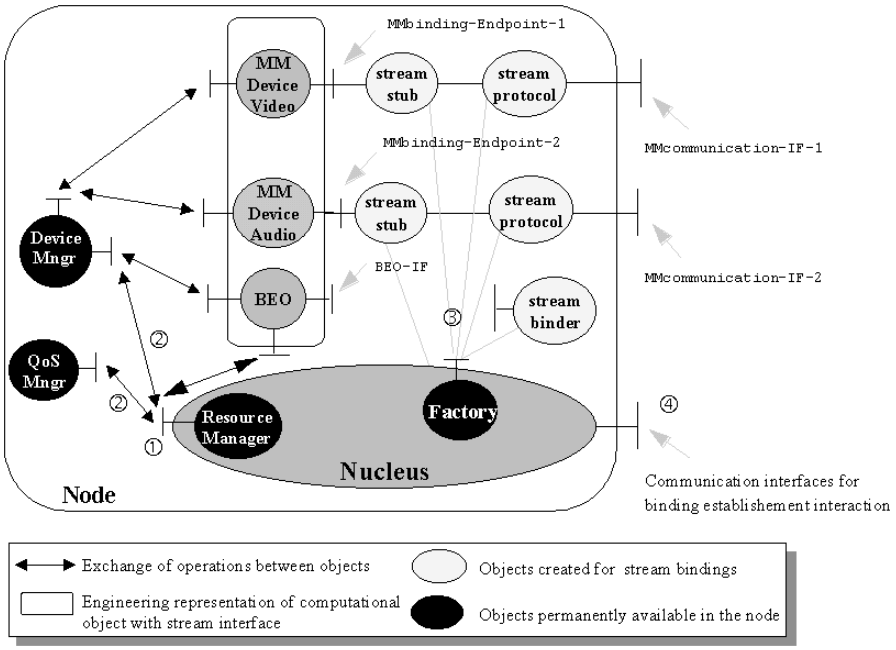



Figure 5.16: Configuration of engineering objects in a node

Using the configuration of Figure 5.16, possible scenario for channel establishment between two BEOs residing in different nodes can be as follows:
One of the BEO invokes a `bind()` operation on its local nucleus. The parameters in the bind-operation are (amongst others):

- BEO's interface reference (Figure 5.16, `BEO-If`);
- Interface reference of the remote BEOs (list of remote `BEO-If`)[14];
- List of stream interface references of Multimedia devices associated to the BEO (`MMBinding-Endpoint1, MMBinding-Endpoint2`)[15].

The nuclei (local and remote) that receive the bind request, determine whether the requested binding can be instantiated. Related to the QoS phases [54] they will enter the *prediction* and *establishment QoS* phase. Afterwards, they will return the result of the bind request to the BEOs involved in the stream interface binding.

**QoS prediction phase.**
Before establishing the connection and instantiating all the objects necessary for the exchange of time-based media, a QoS allocation test needs to be performed to determine whether the resources necessary for the binding can be allocated. The allocation test will be performed by the nodes, and by underlying networks involved in the binding.

For each node involved, the nucleus will invoke a QoS allocation test operation (Figure 5.16, ①) on its Resource Manager to determine whether the QoS characteristics can be met. The resource manager has an overview of the current load of the node, and has the mechanisms to determine whether the requested QoS characteristics as described in the environment contract can be accepted. This test is made in order not to violate established QoS agreements of other processes that are active, or to overload its processor. Interaction between the Device manager and QoS manager will occur to verify the allocation request (Figure 5.16, ②). The result of the allocation test contains a description of the values of the QoS parameters for which allocation can be guaranteed.

The result of the allocation tests will be analysed either by one of the nucleus or by a third party. The QoS establishment phase is entered when the allocation test has been passed successfully.

14 In this scenario it is assumed that the interface references of the remote BEO are obtained via a trader or previous interactions.

15 It is also assumed that upon deployment of the computational multimedia object, the BEO knows the stream interface references of its associated MMdevices.

### QoS establishment phase

Communication between objects located in different nodes is possible if the necessary objects are instantiated, the QoS between the parties agreed, and the stream interfaces (i.e. `MMcommunication-IF`) are bound.

The `resource-allocation` operation allocates the resources and instantiates the stub, binder and protocol objects in the nodes (Figure 5.16, ②). For a 'compulsory QoS' agreement, the resource manager will allocate the appropriate resources such as the MM devices and a certain processing capacity for the BEO to guarantee the resource availability.

QoS negotiation between nodes is possible (a third party can be involved as well). Depending on the guarantee level, various QoS mechanisms are possible [54]. In a compulsory QoS scenario neither the provider (third party) nor the responding user is allowed to decrease the quality as supplied by the initiating user. However, the QoS compulsory value can be increased by the responding user to a value that does not exceed a certain bound.

In an ODP system, the initiating user communicates directly with the responding user over the communication interface (see Figure 2.12 and Figure 5.16, ④). In Figure 5.16, a third party provider (e.g. network operator) is not directly involved in the negotiation process but indirectly through the protocol object and its associated QoS characteristics that have to be met in a compulsory QoS agreement.
If QoS agreement has been reached the responding nuclei send their `MMcommunication-IF` references to the initiating user (or a third party) that will perform the binding. The `Bind()` operation will establish the binding between all `MMcommunication-IF` interfaces. The parameter `ConnectionGraph` in the `Bind()` operation describes precisely which flows described in the `MMcommunication-IF` need to be inter-connected.

### QoS operational phase

In this phase QoS monitoring, QoS maintenance and/or QoS enquiry is performed [54]. Figure 5.17 shows the objects that are involved when a channel has been established between two stream interfaces. Local QoS monitoring can be performed by the *QoS node manager* object available in each node. The QoS node manager can invoke QoS operations on the various channel objects (Figure 5.17, ②③④) to adjust the QoS values to maintain the QoS agreement. During the operational phase the QoS manager receives periodically (e.g. every 500 msec) the actual QoS values from the relevant objects and calculates the required system performance for the agreed QoS. It calculates the difference between the required system performance and the measured system performance and invokes the appropriate action (e.g. Figure 5.17, ②④) to achieve the required system performance. The precise mechanisms for QoS tuning are outside the scope of this thesis.

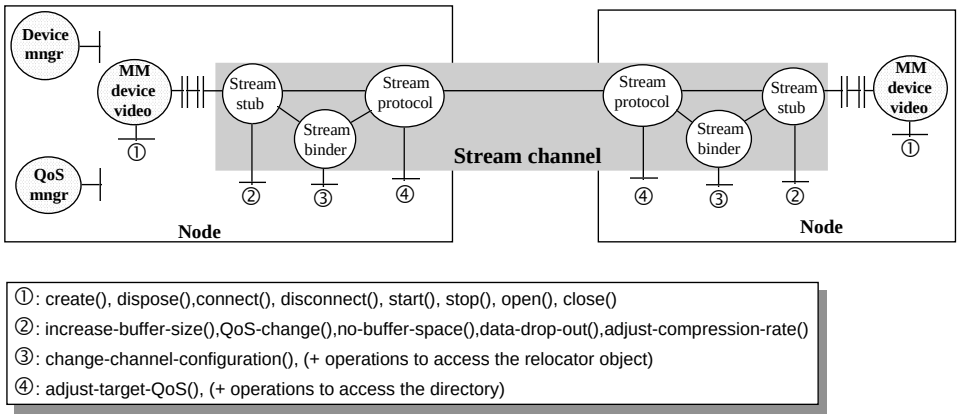



Figure 5.17: QoS operations related to objects in the channel

## 5.4 Relation to ATM technology

The previous sections related QoS to the ODP framework, this resulted in (abstract) models such as Figure 5.17. An interesting question is how to relate these ODP models and QoS to a certain technology. This section gives an example of a possible correspondence between engineering objects and a specific technology, i.e. ATM. The author focuses on the stream protocol object and relates the rather abstract QoS characteristics described in the ISO-QoS framework [54] to several more concrete ATM QoS parameters.

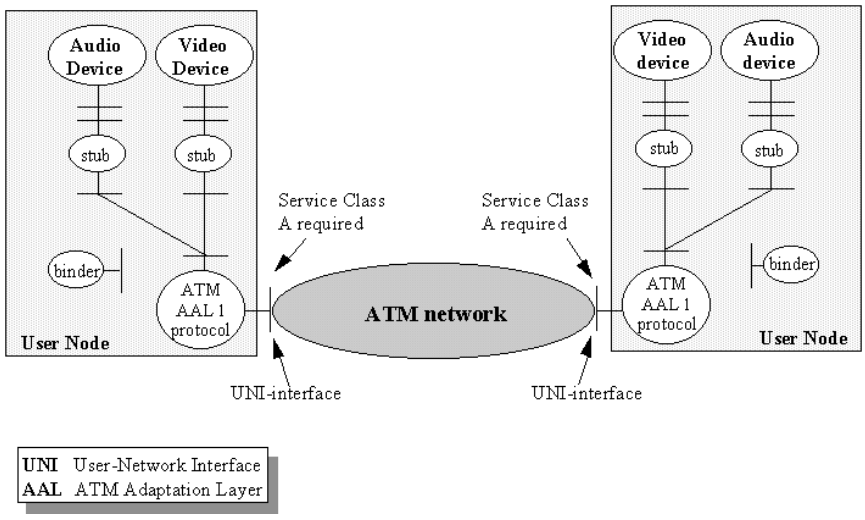



Figure 5.18: Example of ATM AAL1 protocol object

Figure 5.18 depicts a channel showing an end user node supporting the ATM AAL1 protocol. For the support of Constant Bit-Rate (CBR) audio and video, the so-called *Service class A* is the appropriate QoS class [101][102]. Service class A is connection oriented and a strong timing relation between the source and sink is required but the bit rate is constant. QoS parameters in this class consist of cell transfer delay, cell delay variation and cell loss ratio. Data passed from the audio and video device in Figure 5.18 is passed to the ATM AAL1 protocol at the source at fixed intervals and must be passed to the sink at the same rate.

If the user requests for a certain service class (e.g. service class A) this implies that certain ATM QoS parameters are set for the AAL1 protocol (Table 5.8, right column).

| QoS characteristics (QoS-framework) | ATM QoS parameters |
|---|---|
| Time related characteristic | • Cell interarrival time<br>• Cell Delay Variation (delay jitter)<br>• Cell Transfer Delay |
| Capacity related characteristics (communication throughput) | • Cell Transfer Capability<br>• Mean Cell Rate |
| Guarantee level | • Best effort class (unspecified QoS Class)<br>• Statistical/compulsory class (specified QoS class) |
| Channel availability (the proportion of time that the communications channel is available) | • Priority Level (loss priority level )<br>• Cell Loss probability |
| Accuracy (Error related parameters) | • Cell Sequence Integrity<br>• Cell loss Ratio<br>• Cell Error Ratio<br>• Cell Misinsertion Rate<br>• Severely-Errored Cell Block Ratio<br>• Cell-insertion rate (number of cells in a certain time interval that are inserted (from other connections) in a connection.<br>• Bit-error probability (in the information field) |

Table 5.8: Translation of QoS parameters onto ATM QoS parameters

Table 5.8 shows that the more general QoS characteristics (left column) as defined in [81] can be refined for ATM QoS parameters (right column). The ATM parameters apply to cell streams only and are intended to characterise ATM connections that can be set up between two nodes as shown in Figure 5.18. The mappings shown in Table 5.8 are not trivial and additional work is needed.

## 5.5 Conclusions

This chapter discussed ODP computational and engineering issues to integrate multimedia in DPEs. These issues are of interest for both industrial consortia (OMG, TINA-C, DAVIC) and standardisation groups (ISO/IEC, ITU-T), which need to deal with multimedia applications that have stringent QoS requirements.

The ODP framework provides concepts to specify QoS characteristics of objects that deal with time-based flows. Currently, the environment contract concept in ODP-RM is not detailed and serves merely as a placeholder. The ideas and solutions presented in this chapter have been proposed to and accepted by the ODP standardisation group and are described in two 'questions' that deal with them:

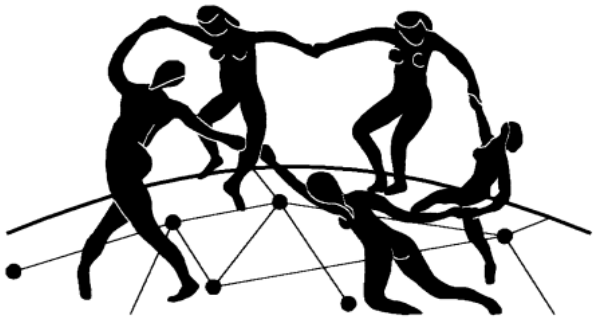
- *'QoS and ODP'* [23]. For this question, this chapter provides answers how the concepts, as defined in the QoS framework [81], can be applied to ODP-RM. Material described in this chapter has been integrated in the draft question answer and part of the New Work Item on 'QoS and ODP' that starts in January 1997;
- OMG is in the process to incorporate several of the concepts presented in this chapter (stream interface, QoS specification). Several concepts presented in this chapter have been proposed as a white paper to OMG's Telecommunication Special Interest Group [88]. The author expects that several ideas and in particular the TINA-ODL specification for stream interfaces will be included in future OMG de-facto standards;
- 'ODP interface references and binding' [55]. This work is now at the level of Committee Draft (CD) and will become a standard that specifies the information content of an interface reference (both stream and operation interfaces). This chapter proposes suggestions how and where to express QoS in the interface reference and refines the rules on typing in the binding process that have been included in the CD document [148].

Requirements were identified that need to be fulfilled by a multimedia binding object. In addition, suggestions for a general multimedia binding object and generic control interfaces have been described using results of de-facto standards as input. The author believes that it will be very hard to fulfil the requirements for binding objects in practice. Instead, it is expected that more specialised multimedia binding objects will be standardised, which provide limited functionality (e.g. a videobridge connecting a maximum of three users, which must have the same flowtypes). Additionally, de-facto standards can be used as a specific implementation of a multimedia binding object (e.g. IMA virtual connection object).

The ISO-QoS framework is used as a basis for the ODP environment contract concept. Because of its general nature the author encountered difficulty to relate the (rather) abstract QoS characteristics to concrete QoS parameters as, for example, defined by ATM. The author believes that the QoS framework could be improved if translations were made to various network technologies.

An open issue is the translation of computational QoS specifications onto engineering characteristics. This requires further study; hints were provided on how to tackle this problem. In general, it remains difficult to relate specifications expressed in different ODP viewpoints. This is also nicely expressed in the following (Erich Gamma et al., Authors Design Patterns (Addison Wesley, 1994):

> *'An object-oriented program's runtime structure (engineering) often bears little resemblance to its code structure (computational). (...) Trying to understand one from the other is like trying to understand the dynamism of living ecosystems from the static taxonomy of plants and animals, and vice versa.'*



# 6 Case Studies

*This chapter presents two case studies that show the practical use of the proposed multimedia extensions and methods to specify distributed services presented in the previous chapters. The case studies differ considerably in scope. The multimedia conferencing service presents a complex service encompassing many objects. The synchronisation service is smaller in scope and is a specific component often present in multimedia services. The same design method is applied to both case studies to illustrate that material described in this thesis can be applied to a broad variety of services ranging from rather small services to complex services.*

## 6.1 Introduction

The design of distributed services remains a challenging task. Many object oriented design methods are available and companies often adopt a particular design method to be used in their (software) projects. IBM studies show that OMT, BOOCH and Objectory are the most used design methods. The IBM study also revealed however, that 60% of all software projects do not use any method at all.

To apply a design method is desirable to keep track of the project progress. Several approaches are possible of which the *waterfall* approach and *iterative* approach [153] are well known. With the waterfall approach, the tasks to come to a software product are performed sequentially. One starts with the analysis and gathers all the requirements at this point. Subsequently the design, coding, testing and integration tasks take place. The waterfall approach assumes that the requirements are known from the analysis task and the other tasks can be defined from that point on. This is often incorrect as all the requirements are rarely known from the beginning and they might also change during

the project progress. The waterfall is characterised that a particular task is finished with a document and one can proceed with the following step without returning to the previous task. The consequence of this approach is that missing pieces (or errors) at the start lead to problems further down the waterfall. With an iterative approach many problems of the waterfall approach are solved. An iterative approach allows jumping back and forth to other tasks in the project life cycle allowing specifications to be incomplete the first time. These specifications can be adapted and extended during the project life cycle. The basic goal of the iterative design approach is to deliver software products in shorter cycles even with fewer clearly defined requirements.

In this chapter it is not the intention to define a new design method. Instead a combination of methods are applied that have been developed by both de-facto and de-jure standardisation bodies. This combination of methods covers the whole life cycle of a telecommunication service. In this chapter the focus is only on the design trajectory of a telecommunication service and aspects such as the installation of a service into an operational environment are disregarded. Two case studies are described which apply the concepts and ideas proposed in the previous chapters. The *multimedia conferencing* case study shows how a distributed telecommunications service can be developed. Distributed multimedia conferencing services are emerging applications that are of interest for both the residential and business environment. Several conferencing services prototypes have already been developed that operate in a heterogeneous workstation environment consisting of different hardware and software platforms. These implementations show that equipment heterogeneity can be solved which is an important factor for a universal availability of distributed telecommunication services. Another important requirement for commercial success is interoperability between conferencing services. Interoperability implies, for example, that users using a specific conferencing service (e.g. JVTOS [131]) would like to work together with users using another conferencing implementation (e.g. BERKOM [132]). To solve this problem of interoperability this chapter proposes generic interface specifications for conferencing services.

Additionally, distributed conferencing services have specific requirements with respect to the exchange of time-based flows. One of these requirements is the synchronisation within and between related time-based flows. To address this issue the second case study specifies a *synchronisation* component to solve synchronisation of time-based flows.

## 6.2 Design method

A combination of design concepts is selected based on the survey and discussion presented in Chapter 3. The TINA life cycle model is used as a framework (i.e. road map) to systematically structure the various specifications that arise when developing a service. Since the TINA life cycle model lacks details (see also Section 3.5) it is completed with the ODP viewpoint languages, SG15 enterprise template and OMT concepts.

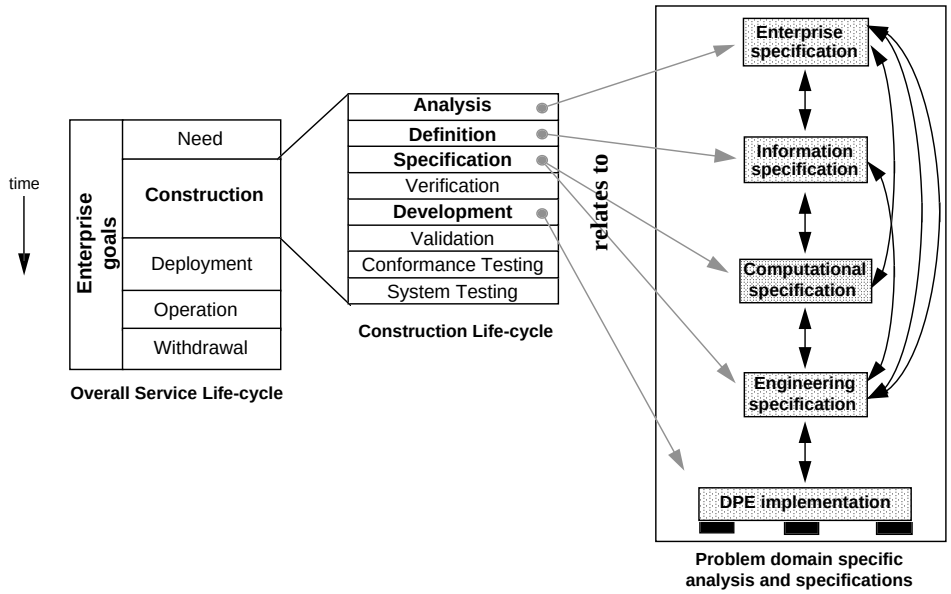



Figure 6.1: TINA life cycle model applied to case studies

The *TINA life cycle model* [63] defines an ordered set of steps that are required to support the development, deployment, and operation of a service. Only the construction life cycle, which is part of the life cycle model is applied in this thesis since the focus is on the design of the services. The *construction life cycle* is apt for this purpose but TINA has not come up with details of what to do in each phase. Therefore, details were added using the ODP viewpoint languages for the construction life cycle, as shown in Figure 6.1. It should be noted that as with many other design methods, the TINA life cycle model is not a strict waterfall model (i.e. not a strict top-down approach) of service development. In each phase of the construction life cycle, it is possible to return to a previous phase if refinements and requirements are added during service development. This is natural since requirements emerge throughout the development of the service.

This means that an iterative approach is applied in which the different task in the construction life cycle (and related ODP viewpoint specifications) are developed in an iterative manner. The ODP framework allows this flexibility (see also discussion in Chapter 2 on the use of ODP viewpoints). It is possible to have, for instance, a detailed computational model before the information model is completed.

The overview of the case studies is presented using computational concepts (starting with the specification phase). The reason for doing this is that experience obtained from several projects shows that the computational viewpoint offers the appropriate abstraction level to communicate the ideas and purpose behind the service to be

developed with others. Based on this overview described in computational terminology the case studies are detailed using the five viewpoint languages[16].

### Construction phase

The construction phase is defined in [63] as 'all the off-line activities required in designing and developing the software and any special hardware associated with a service'. The construction life cycle is further decomposed in several activities.

- In the *analysis phase*, relevant multimedia service requirements, obligations and policies are identified from the different stakeholders involved. A stakeholder could be the end-user, the service provider or the network operator. The ODP enterprise language is used and complemented with the ITU SG15 template [154] to express the requirements related to the service. The ITU-T study group 15, Question 30 defines a notation (a template) using the ODP-RM part 2 concepts as a basis. The template enables the specification of a system from an 'enterprise' perspective. The formal description in BNF is described in Table 6.1. This template is only applied to the synchronisation case study to test its suitability;

```
<community_template> ::=
   "COMMUNITY" <label> <name>
   <doc_heading> "PURPOSE" <community_ definition>
   <doc_heading> "ROLE" <role_definition>*
   <doc_heading> "POLICY" <policy_definition>*
   <doc_heading> "ACTION" <action_description>*
   <doc_heading> "ACTIVITY" <activity_definitions>*
    ["WITH ACTION GRAPH" "Start" <action label>* "End"]
    "CONTRACTS" <label_string>

<action_description> ::=      <doc-heading> <label> <label_string> "ACTION POLICY"":"
                               <policy_definitions>
<action_ label> ::=           <label>
<activity_definition> ::=     <doc_heading> <label> <label_string>  "ACTIVITY POLICY"
                               <policy_definitions>
<activity_definitions> ::=    "None"¦<activity_definition>*
<community-definition> ::=    <label_string>
<doc_heading> ::=             <text>
<label> ::=                   "[A-Za-z] [A-Za-z0-9.]*"
<label_string> ::=            --quoted string ¦< label>
<name> ::=                    <label>
<policy_definitions> ::=      <none>¦<policy_definition>*
<policy_definition> ::=       {"PERMISSION" ¦"OBLIGATION"  ¦"PROHIBITION"
                               ¦"EXCEPTION"} <label> <label_string>
<role_definition> ::=         <label> <label_string>
```

Table 6.1: Template for the enterprise language

- The *definition phase* provides a description of the desired effect of the multimedia service as seen by the stakeholders. The description specifies ***what*** the service will do and not ***how*** it will be implemented. The ODP information language (and in

16 Note that for readability purposes, the results are presented starting with the enterprise specification up to the technology specification.

particular the OMT notation) is suitable for describing the information aspects related to the service independent from any implementation;

- The *specification phase* provides a formal and unambiguous description of the service. It should include a specification of how the service can be implemented. For the specification of the case studies the computational and engineering concepts as defined in ODP-RM are applied. Additionally, the TINA concept of grouping and the OSCA three layer approach (see Section 3.5.1) are useful to classify the computational objects;
- In the *verification phase* the equivalence between the service specification (as a result of the specification phase) and initial requirements (result of the analysis/definition phase) are studied. In case of differences adaptations have to be made;
- The *development phase* comprises the development of the software and hardware modules needed for the multimedia service;
- The *validation phase* consists of verifying the developed software and possible hardware modules in the development phase against the (paper) specifications of the specification phase. The validation phase can be subdivided in the *conformance testing* phase and *conformance testing* phase. The conformance testing phase includes the checking of the implementation for conformance to architectural rules and standards used in the design. Thus, it checks the conformity between the output of the development phase versus definition phase versus specification phase. The system testing phase comprises the testing of software and hardware modules in a test environment, which represents an operational environment. When successful the service can be deployed in an operational environment.

## 6.3 Case 1: Multimedia conferencing service

Multimedia conferencing offers integrated support from the desktop for audio-visual information exchange, conference management and sharing of applications such as word-processing and whiteboards. It provides powerful mechanisms for remote interaction and supports various communication schemes between users. In many Research and Development environments advanced application scenarios for multimedia collaboration can be observed. Joint-viewing and tele-operation on documents, computer programs and simulations are being demonstrated [131][132].

### 6.3.1 Overview

Functionalities provided by multimedia conferencing services are extensively described in the literature (e.g. [126][127] [131][132][133][135]). These services haven often similar and common functionalities, which characterise a conferencing service. Figure 6.2 shows the most typical conferencing functionalities grouped in computational objects and their interfaces.

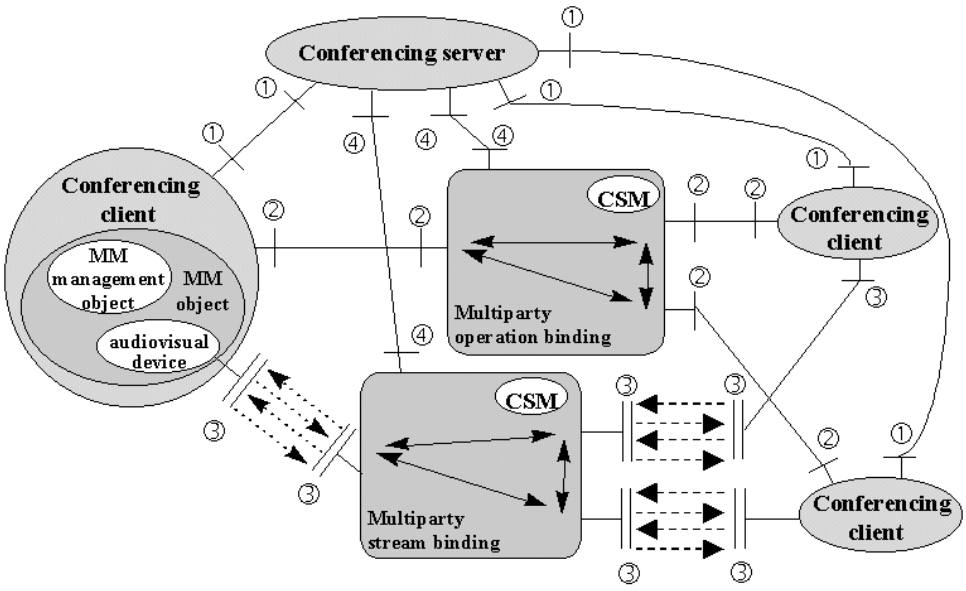



Figure 6.2: Overview of multimedia conferencing service

**Multiparty bindings**

The *multiparty stream binding* and the *multiparty operation binding* objects are a specialisation of the binding object described in Section 5.2.3. The two binding objects

in Figure 6.2 are quite similar except that one object handles time-based flows and the other one handles discrete operations. Control operations (④) can be performed on the multiparty binding object. In TINA, those operations are received and processed by a CSM object, which is responsible for the end-to-end connectivity (see Section 4.3). Its control interfaces (④) receive operations such as add new interface, change QoS etc.

The *multiparty stream binding* object manages the interactions between the stream interfaces (③). It has a control interface through which operations are provided for control purposes. An ODP specification of the stream binding for multiparty audio and video flows can be found in [21].

The *multiparty operation binding* object manages the multiparty operations invoked on the shared applications. It receives all operations (②) of the involved participants and rebounds them (after possible manipulation) to the clients. It has a control interface (④) that receives operations from the server such as add new client and change floor control.

### Conferencing client

The conferencing client object is a so-called TINA object group, (see Section 4.3) and contains, amongst others, the *multimedia object* as defined in Section 5.3. The conferencing client has three generic external interfaces:

- *Control & management interface* (①) used to support operations, which deal with the control & management of the conferencing service;
- *Functional interface* (②) used to support the functionalities provided by the conferencing service (e.g. telepointer);
- *Audiovisual interface* (③) used for the exchange of time-based flows between all conferencing clients but also between a single conferencing client and the conferencing server (e.g. video to be stored in the multimedia conferencing database).

### Conferencing server

The conferencing server creates and manages conference sessions. It includes operations that control the flow of actions in a conference session. Furthermore, it administrates conferences and provides facilities to control the assignment of roles to conference participants. This object provides the mechanisms to support many operations of the generic control & management interface (①).

The conference server object is also a TINA object group containing various computational objects (not shown in Figure 6.2) including:

- The *Information base* object models a database containing conference related information about users and user groups. It deals with static (e.g. user names) and dynamic information (e.g. who has the floor);

- The *QoS manager* object manages end-to-end QoS between conferencing server and conferencing clients. The conferencing clients negotiate certain QoS parameters of the conferencing session, which is reflected in a 'QoS contract' between conferencing server and conferencing client. This contract specifies, for instance, allowed audio sample rate, video frame rate and cost per minute. The QoS manager is in charge of controlling the negotiated QoS and to perform corrective actions when necessary. This object provides the mechanisms for the QoS operations of the control & management interface template;
- The *accounting manager* object is responsible for the end-to-end billing. It uses the so-called one-stop-shopping principle which means that the conferencing client gets one bill including the costs for using the transport capabilities provided by the network operator. The accounting manager collects billing information from the network operator and add to it the costs of the conferencing service;
- The *synchronisation manager* object manages synchronisation that are required for multimedia conferencing services (see also the synchronisation case study).

### 6.3.2 Analysis phase (enterprise specification)

The enterprise specification provides a description of the requirements and constraints that the environment imposes on the multimedia conferencing. The ODP enterprise viewpoint terms of *artefact*, *agent* and *actions* are used to describe the service. In addition to these concepts, the concept of role is refined, which can be played by a stakeholder with respect to a service. The focus is on the end-user, service provider and network operator roles.

#### End-user

The end-user defines the service needed and the corresponding values of QoS parameters. With respect to flows, QoS parameters specify audio and video quality in terms of broadcast TV quality, HDTV, telephone audio quality, Hi-Fi or CD quality. Furthermore, QoS requirements between flows are defined. For example, the lip-synchronisation between audio and video.

#### Service provider

The service provider offers the multimedia conferencing service taking into account the end-user's policies, which it has to support and to manage. It supports *actions* to create and delete end-users or to adjust policies of existing end-users. The service provider performs end-to-end (re)negotiation with end-users to determine preferred, acceptable, and unacceptable values of QoS parameters. The latter is reflected in the 'conferencing service contract', which is a multimedia service contract that describes the result of the negotiation between the end-user, service provider and network operator. The service provider can set up, remove, and adjust the service in accordance with the (re)negotiated end-to-end QoS requirements.

### Network operator

The network operator provides end-to-end connectivity for the conferencing service. It is an *agent* that manages bindings in correspondence to the status of the supporting network. The network operator is responsible for the flow topology (point-to-point, multipoint-to-multipoint etc.), billing, security, fault management, and QoS provided by the underlying network and resources. It manages the network so that the multimedia conferencing contract is guaranteed. For instance, it selects an appropriate routing for the audio and video channel and reserves resources on each node on that route.

## 6.3.3 Definition phase (information specification)

The information viewpoint specification of the multimedia conferencing service describes the information relevant to the service. It takes into account the requirements and objectives outlined in the analysis specification. Information which is relevant to both end-users, service providers, and network operators is specified in the conferencing service contract. This contract is the outcome of an agreement between the stakeholders. It satisfies their requirements and objectives as laid down in the enterprise specification. Figure 6.3 shows the relation between enterprise and information specification in terms of OMT.

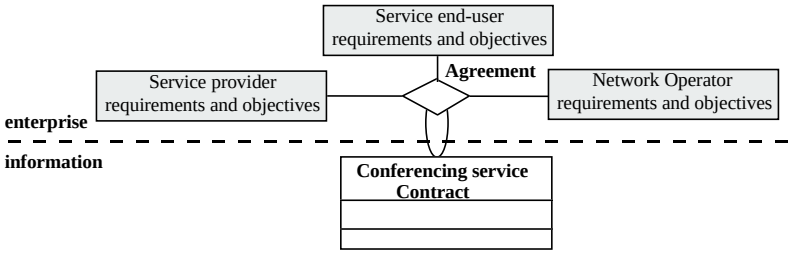



Figure 6.3: Contract between the stakeholders

The conferencing service contract, is here described as a single class. However, at a more detailed level, the information specification is more complex. This case study focuses on a subpart of this contract: the multiparty stream binding contract.

### Invariant schema of the binding contract

The common structure of contracts between the stakeholders is specified as an invariant schema. Figure 6.4 shows the invariant schema of the *multiparty stream binding contract*, which describes a part of the conferencing service contract defined previously . The contract contains information about the end-users in the binding (user information objects), the stream interfaces involved in the binding (stream interface information objects) and the operations that stakeholders can invoke. Additionally, the contract specifies the stream binding between stream interfaces. The information in the binding is modelled by means of the multiparty stream binding information object. The invariant schema for this object specifies all the operations that stakeholders can invoke.

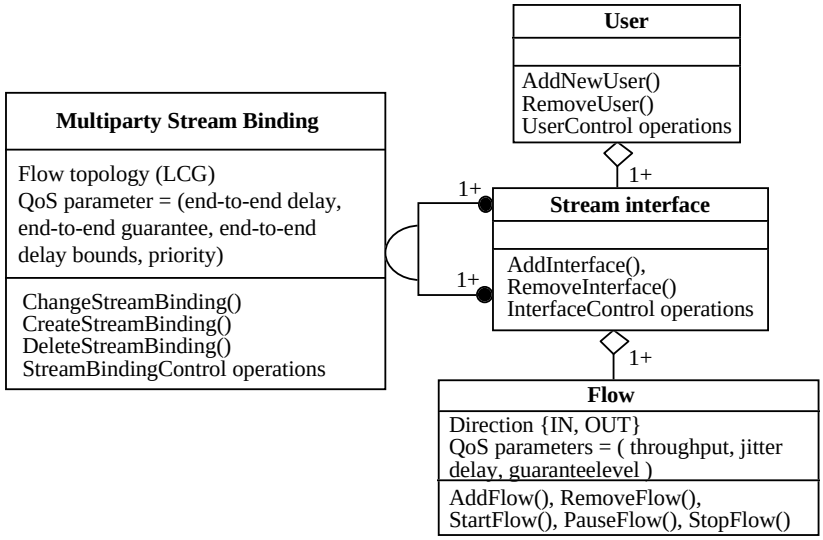



Figure 6.4: Invariant schema of multiparty stream binding contract.

A user may have one or more stream interfaces (e.g. if he participates to two parallel conferences), therefore each user information object consists of one or more stream interface information objects. A stream interface consists of one or more flows, which results in a stream interface information object consisting of one or more (audio, video, or composite) flow[17] information objects. A flow information object as described in Section 5.2.1, consists of attributes indicating, amongst others, the direction of the flows and QoS parameters. The QoS parameters defined in the information specification can be manipulated by computational operations.

Information about the binding is captured by the multiparty stream binding information object. This information object relates two or more stream interface information objects. It contains information about the flow topology and specifies the QoS that needs to be maintained while exchanging audio and video flows between interfaces.

### Dynamic schema of the binding contract

The dynamic schema of a binding contract defines the effects that invoked operations have on the contract and the conditions under which these operations can be invoked by the stakeholders.

The *effect of operations* on the contract heavily depends on implementation choices. In general the effects can be classified into 3 categories: notification effect, negotiation effect, and no effect.

---

17 The term ‘flow’ in the information viewpoint is similar to the TINA term ‘flow endpoint’. However, the term flow is used here to be consistent with ODP-RM standard and Chapter 5.

An operation having a *notification effect* is an operation by which a stakeholder informs the binding contract of a newly created computational object. For example, the operation `AddnewUser` results in a new user information object, one or more stream interface information objects, and one or more flow information objects.

An operation having a *negotiation effect* is an operation by which a stakeholder negotiates a change in the binding contract with the binding object and other stakeholders. If negotiation is successful, changes are made to the binding contract. For example, a successful `RemoveUser` operation results in the removal of a user information object and related stream interface and flow information objects. Furthermore, the binding information object that relates the interfaces of the removed user to the interfaces of other users will be modified or removed.

An operation having *no effect* is an operation that does not affect the information objects in the binding contract. These operations are mainly control operations. For example the operation `PauseFlow` will have the effect that the concerned flow is paused. This does not have consequences for the binding contract information.

The dynamic schema also describes conditions for invoking operations on the binding contract. The invariant and static schema do not impose a specific ordering of operations that stakeholders can invoke. However, to obtain a meaningful binding contract, it is necessary to define conditions with respect to the invocation of operations. For example:

- A customer can only invoke an `AddNewUser` operation on an existing stream binding;
- Customers can only invoke `ChangeAudioQosFlow` and `RemoveFlow` on existing flows;
- Customers and providers can only invoke a `ChangeStreamBinding` operation or a `DeleteStreamBinding` on an existing binding.

### 6.3.4 Specification phase (computational specification)

The correspondence between information objects and computational objects is not necessarily one-to-one. The correspondence between the information and computational specifications must be specified so that consistency between both specifications can be proved. Grouping of classes into objects is a decision taken by the service designer and distribution aspects need not be taken into account at this stage.

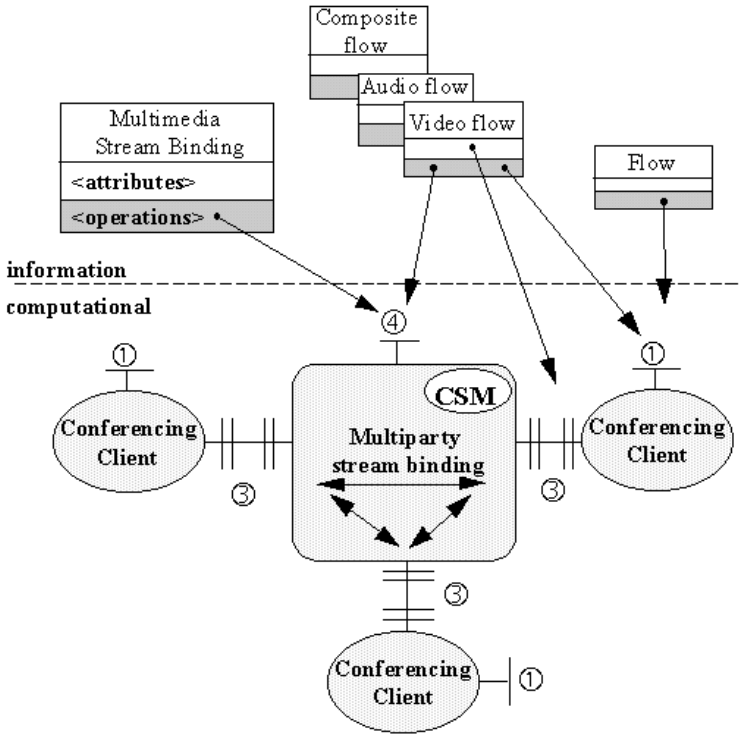



Figure 6.5: Correspondence between information and computational objects

The arrows in Figure 6.5 show the correspondence of several information classes of Figure 6.4 to computational interfaces. The operations specified in the information `multiparty stream binding` class are used for the specification of the operations of the computational multiparty stream binding control interface (④). The management interface of the multimedia object (①) will be reflected in the operations defined in the `Flow` class. The audiovisual interface (③) has the characteristics of the attributes of the information `Flow` class. The attributes specified in the information specification will be reflected into parameters in computational operations. These relations will be shown in the following sections.

## Conferencing client interfaces

Specification of generic conferencing interface templates ensures application interoperability when these templates are used implementations of conferencing services. Therefore the focus is on the interface descriptions of the multimedia conferencing service which should be supported by both the server and client. The functionalities provided to the client can be classified according to three generic interfaces, as shown in Figure 6.6.

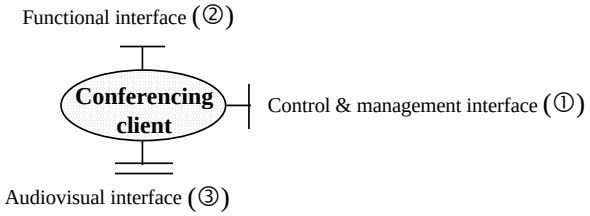



Figure 6.6: Conferencing client interfaces

The *Control & management interface* (①) is used to support control & management operations for the conferencing service. The operations described in this template are exchanged between the conferencing service and the conferencing clients. Table 6.2 classifies the operations into several templates representing a logical grouping of functionalities offered by the control & management interface.

| Interface template | Operations |
|---|---|
| Administration of conferences | Conference (register, unregister, list,...)<br>Group(register, change, unregister, list,...)<br>Policies(user roles, token, QoS,...)<br>User(register, change, unregister, list,...) |
| Conference negotiation | user role, token, QoS,... |
| Conference session | open, close, assign-chair, query, new-chair,... |
| Conference participation | join, leave, new-user, user-left,<br>join_request,... |
| User role administration | change, list,... |
| Token passing | request, revoke, release, list, assign, ... |
| Application sharing control | share, unshare, query,... |
| Flow control | remove, add, set audio video mixing, synchronise-play-out, ... |

Table 6.2: Interface templates for control & management interface

The operations defined in the various templates of Table 6.2 are used to initialise and control a conferencing session. Operations to register users, and policies can be negotiated among the conferencing clients, such as token policies and user roles. Users can join or leave ongoing conferencing sessions and facilities are provided to control shared applications. For the audio-visual communication, operations are defined that allow to add and remove flows.

**interface template** Conference-Session; /* template of control & management interface (①) */

**typedef** .... UserInfo; /* structure containing info about a conferencing client */
**typedef** .... Tokenpolicy; /* description of the possible token policies */

**operations**
OpenConference(**in** UserInfo participants, **in** Tokenpolicy token, **in** UserRoles users);
CloseConference(**in** ConferenceId confId);
AssignChair(**in** UserInfo NewChair, **out** Status result);
Query(**in** ConferenceId confId, **out** ConferenceInfo, info);

**behaviour** 'an instance of this interface template contains operations to control the conference session.'

Table 6.3: TINA-ODL specification of control & management interface.

The *Functional interface* (②) is used to support the operations provided by the conferencing server (e.g. telepointer functionality). This interface template describes the functionalities not dealing with control & management or streams. Operations related to shared applications (e.g. cut & paste actions for a joint editor) are an important group described in this template. The description of this template depends on the applications that can be shared between the clients.

These shared applications relate to the concept of workspace [127]. A workspace contains objects that are created and manipulated by one or more users. The behaviour of the shared application depends on its workspace status. The workspace can be either *common*, which implies that all conference actions are visible among all users or *closed*, which indicates that the conferencing service offers sub conference services to certain users. The workspace can also be *local*, which indicates that conferencing clients do not share information.

| Interface template | Operations |
|---|---|
| Shared application name<br>(e.g. Telepointer, Joint editor) | actions of the application<br>(e.g. move telepointer, cut & paste text) |

Table 6.4: Interface template for functional interface

The telepointer application is an illustration of a shared application. Operations such as move-pointer are described in the interface template together with the behaviour of the telepointer application.

The *audiovisual interface* (③) is used for the exchange of time-based flows between the users. It is also used between one user and the conferencing service (e.g. to store video in the multimedia conferencing database) but this interface is not shown in the figures. The Stream Interface template consists of several type definitions of time-based flows as depicted in Chapter 5, Table 5.2.

### Computational choice for the multiparty stream binding

The multiparty stream binding can be refined (using decomposition). Several implementations of distributed multiparty systems have a functional component,

referred to as *stream controller & dispatcher*, which manages the flow [127][135]. It receives all the flows of the producers and rebounds the flows (after possible manipulation) to all consumers. This approach is adopted for the multiparty stream binding as illustrated in Figure 6.7.

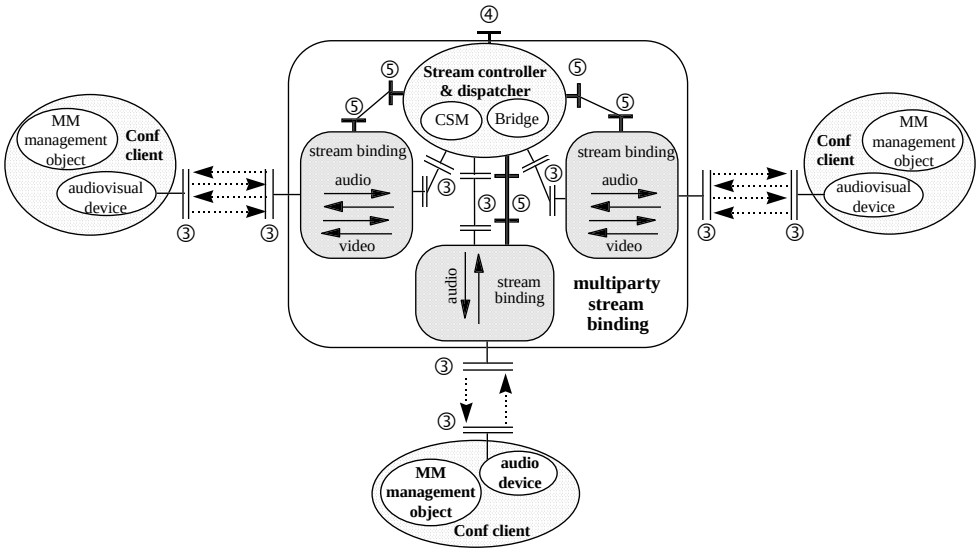



Figure 6.7: Example of objects involved in a multiparty stream exchange

The stream controller & dispatcher object is in charge of redirecting stream control operations (④) to each sub-stream binding (⑤). This object also deals with the set up, control and release of audio and video bindings between producers and consumers.

A partial TINA-ODL specification of a multiparty stream binding object is shown in Table 6.5.

```
object template MultipartyStreamBindingObject;

typedef AVuserId ..../* This type identifies uniquely the audio and video users involved */
/* Each flow has a direction In or Out */

typedef struct {...} videoFlow;  /* see Chapter 5, Table 5.2 for definitions */
typedef struct {...} audioFlow;
typedef struct {...} combinedFlow;
typedef struct {...} Flow;

struct {
   AVuserId  ProducingId;
   AVuserId  ConsumingId;
   Flow      ConcernedFlow;
   integer   NumberOfFlow;          /* Unique identifier of flow */
} FlowBindingId;

typedef sequence <FlowBindingId> StreamBindingId;

initialisation
void init (out StreamBindingControlInterface StrmCntrlInterf)

supported interfaces
   StreamBindingControlInterface = StrmCntrlInterf;

behaviour
'An Instance of this object binds two or more stream interfaces'
```

Table 6.5: TINA-ODL specification of MultiPartyStreamBindingObject

Table 6.6 shows the partial TINA-ODL specification of the stream binding control interface.

```
interface template StreamBindingControlInterface; /* operation interface type (④) */

typedef sequence < Flow> StreamInterface;

operations
void ChangeQosStreamBinding (in StreamBindingId Binding,
                             in QoS RequestedQos, out QoS ProvidedQos);

void RemoveStreamBinding (in StreamBindingId Binding,
                          out StreamBindingId RemainingBindings);

void AddNewUser (in AVuserId Newuser, in StreamInterface NewFlows,
          in QoS RequestedQos,  out QoS ProvidedQos, out ResultReport StatusBinding);

/* Additional stream binding control operations are possible determined by the application, system and
network management part. */

behaviour
'An instance of this interface template provides other objects to perform control actions on the Multiparty
stream binding object.'
```

Table 6.6: TINA-ODL specification of StreamBindingControlInterface.

### 6.3.5 Specification phase (engineering specification)

The computational specification should correspond to an engineering specification to be executed. This engineering specification preserves the behaviour described in the computational specification. For the multimedia conferencing, the focus is in the engineering specification on the channel configurations as shown in Figure 6.8.

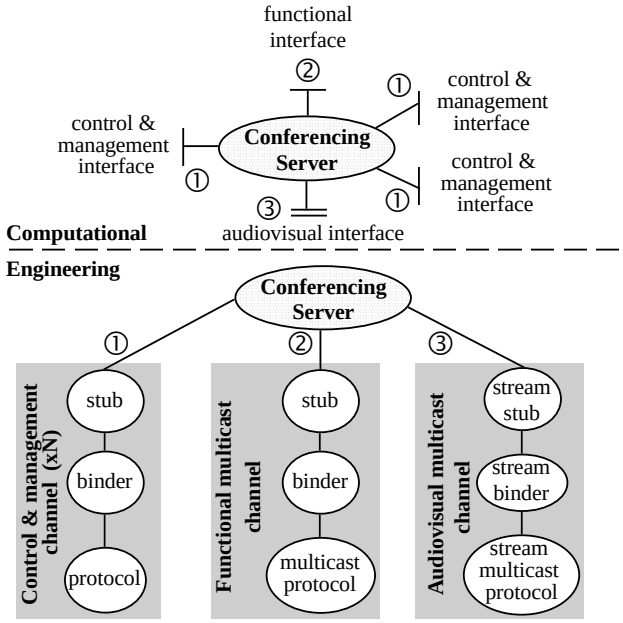



Figure 6.8: Correspondence between computational interfaces and engineering channels

A *Server control & management interface* (Figure 6.8, ①) is reflected in the engineering specification as a control & management channel. The environment constraints specific for the interfaces (e.g. security constraints) are taken into account while establishing the channels between the objects involved. A control and management channel conveys the control and management operations (e.g. floor control, QoS negotiation). The stub object provides marshalling/unmarshalling of operation parameters to enable access transparent interactions. The protocol object assures that computational objects can interact remotely with each other.

The *Functional interface* (Figure 6.8, ②) is reflected in the engineering specification as multiple client-server channels. The environment constraints specific for the interfaces are taken into account while establishing a channel between the concerned objects. The channel should support both low traffic volume and bulk data transfers that need high throughput and should be reliable (e.g. distribution of graphics and conferencing minutes). In case of bulk transfer visible delays are tolerable. The channel should also support non-real-time interactions (e.g. text updates) that need low traffic volume but

high reliability. RSVP and XTP [129] are two lightweight multicast transport protocols of interest for the engineering protocol object of this channel.

The engineering representation of the computational *Audiovisual interface* (Figure 6.8, ③) leads to the creation of one multicast producer/consumers audiovisual channels specialised for time-based flows. The QoS parameters associated with the flow defined in the computational specification, influence the choice of the audiovisual multicast channel components. The *audiovisual multicast channel* transports real-time interactions, which need high throughput, bounded delay jitter, but tolerate transmission errors. Streams require different functionality of the stream stub object due to the different nature of information that is exchanged. The stub object provides the mechanisms to encode and decode video/audio information. Furthermore, data available for the consumers should be notified, and the stream stub object provides operations to control local resources (e.g. increase buffer-size). Notifications of events concerning the stream are supported (e.g. QoS change, no buffer space available, data drop out). For the exchange of time-based flows, a stream multicast protocol without the RPC mechanism is necessary as discussed in Section 5.3.3. XTP is a multicast transport protocol of interest for the protocol object of this channel.

Figure 6.9 shows how a particular computational specification corresponds to an engineering specification in the context of the TINA architecture. Specific objects appear such as the CSM, TCSM and CC objects (see also Section 4.3.2).

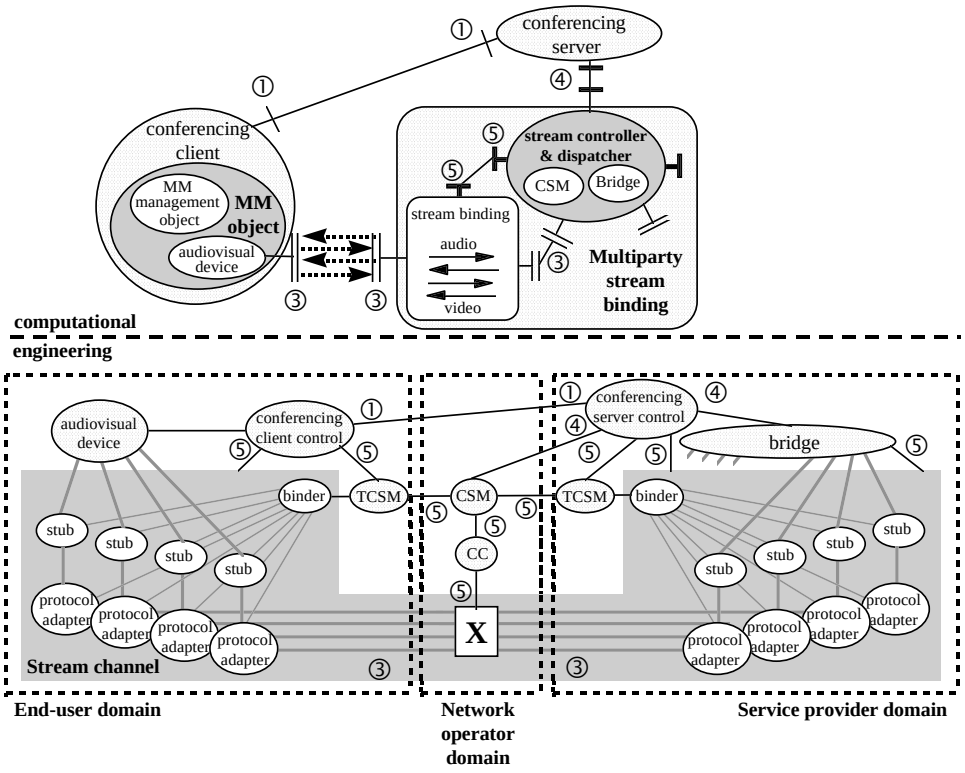



Figure 6.9: Engineering (telecommunication-oriented) solution for time-based flows

The binding object corresponds to the engineering configuration of objects as defined by the TINA connection management architecture [66]. The stream interfaces corresponds to a configuration of engineering objects as proposed in Section 6.2. The application designer dealing with stream interfaces is only concerned with the engineering configuration for a user DPE node. The application designer expects that the service provided by CSM (located in the network operator domain) is available. On the other hand, the network application designer will consider those parts that relate to the objects located in the network operator domain, see Figure 6.9.

## Behaviour specification

Figure 6.10 shows a simplified interaction model[18] for the set up of a conference between three clients. Initially, `client-A` requests the trader for a reference of one or more conferencing services that satisfies `client-A`'s needs(①). The requirements are expressed for example as 'desired QoS of audio/video connections', 'maximum price per minute' etc. The trader will respond with a list of references that satisfy the request

18 This interaction model should be part of a computational specification but is presented here to show the relationship with TINA connection establishment, which is positioned, in the engineering viewpoint.

(②). `Client-A` will then choose a specific conferencing service and contacts the conferencing server (③). If `client-A` and the conferencing server enter into agreement, a confirmation is sent to `client-`A (④). In this scenario, the conferencing server contacts `client A,B,C` for the establishment of a three party conference (⑤). If the clients agree they will join the conference (⑥).

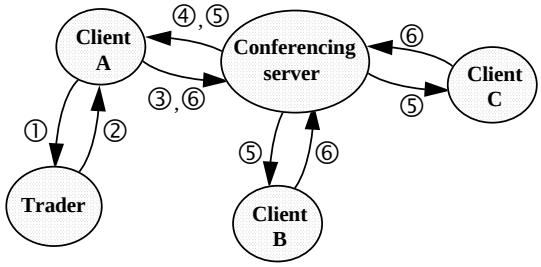



Figure 6.10: Interaction model to initiate a conferencing service

For the actual connection establishment, Figure 6.11 shows a possible sequence using the connection graphs of TINA. In this scenario the conferencing server control object (i.e. a particular instance of the SSM object) sends a LCG-request operation to the CSM-object indicating a three-party binding between the conference bridge and the three clients[19]. The CSM will then decompose the requested binding (i.e. LCG) into two parts: the nodal part (i.e. NCG) and transport network part (i.e. PCG). The CSM object requests the TCSM objects of the clients to take care of the nodal part while the CC object is responsible for the transport network part. Thus, the CSM object requests the TCSM object of each client to set up a binding between the audiovisual device objects and the network access point of the end-user node. Each TCSM object returns a NCG reference operation indicating the result of the binding and additional information regarding the internal binding. Finally, a LCG reference operation is sent to the conferencing server control object indicating the interface reference of the CSM object to manipulate the binding, as well as, additional information such as status information. The conferencing server control object can then set up (activate) the conference bridge and communication between the clients is ready for use.

19 The conferencing server control object knows the addresses of the clients due to the prior interaction as explained in Figure 6.10.

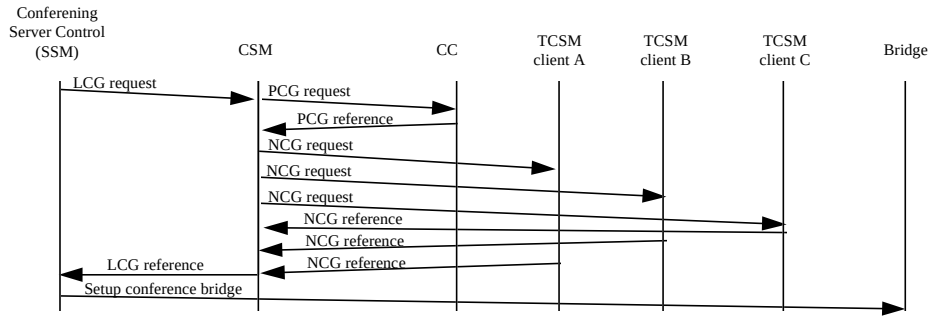



Figure 6.11 : Message trace diagram for connecting multimedia conferencing

### 6.3.6 Development phase (technology specification)

The technology specification consists of a description of hardware and software that implements the engineering specification taking additional enterprise requirements into account (e.g. which hardware and software is available).

At KPN Research a stream binding is implemented using ANSAware 4.0 [124] [140]. It corresponds to an implementation of a multiparty video phone service, which enables end-users to exchange audio-visual information via their desktop computer. Users can be added or removed dynamically.

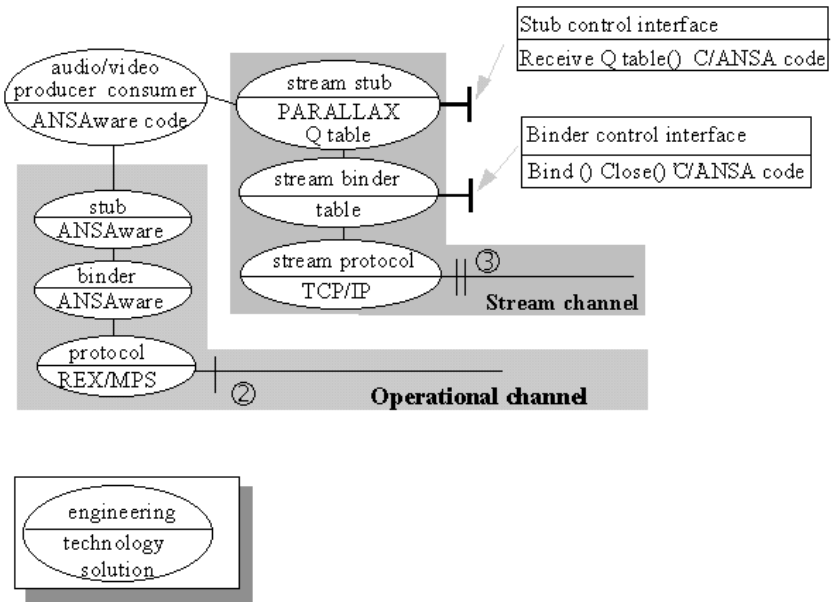



Figure 6.12: Correspondence between engineering objects and specific technology

Figure 6.12 indicates a correspondence between engineering objects and technological solutions. It focuses on the implementation of the stream channel and operation channel and shows how the engineering objects are realised in hardware and software components. ANSAware 4.0 is used as an implementation of the DPE.

The implementation of the *operation channel* is fully supported by ANSAware 4.0. The stub and binder object are generated automatically using the ANSAware tools. The protocol object is refined into three layers of abstraction as shown in Figure 6.13. The highest level is the interface of the interpreter module followed by the interface of the execution protocol and at the lowest level the Message Passing Service (MPS). The MPS service just sends messages contained in a buffer to an endpoint. In ANSAware 4.0, message passing services are implemented on top of Interprocess Communication (IPC), User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). All message passing services have the same interface but differ in the service provided. For example, the MPS based on IPC can only be used for communication between capsules residing on the same node. The execution protocol adds extra functionality to the MPS. This includes fragmenting of messages and retransmission in case of message loss. Furthermore, this protocol ensures that invocations are delivered at the right interface. ANSAware 4.0 supports the Remote Execution Protocol (REX). This protocol is an implementation of the RPC mechanism. The interpreter integrates the execution protocol with the scheduling of threads.

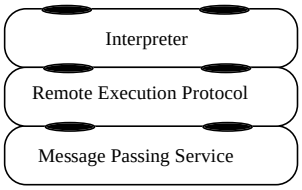



Figure 6.13: Refinement of the operation protocol object based on ANSAware 4.0

For the implementation of the *stream channel* ANSAware 4.0 has been extended with a number of functions to support time-based data. Based on the existing hardware[20] and software to exchange compressed video, a specific implementation is given to the stream stub and stream binder object.

The stream stub object maintains a Parallax quantisation table (Q-Table). A Q-table contains data for the Parallax video cards with respect to the compression of video images which is needed to display video images. A Q-table is created by a producer of video images. The control interface of the stream stub object is used to exchange Q-tables.

The stream binder object is responsible for the mapping of dataflow identifiers onto an endpoint identifier (sockets) that is created by the stream protocol object.

20 Sparc workstations extended with Parallax video cards for the compression and displaying of video images.

The stream protocol object is based on TCP/IP due to the technology available for this implementation. Other transport protocols, more appropriate for real-time time-based dataflows should be used in future implementations. ANSAware 4.0 supports only remote operations by means of a RPC mechanism. For the stream protocol, the RPC mechanism is bypassed and operations invoked on the stream protocol object interface are directly passed to the TCP protocol. The stream protocol provides control operations to create, connect and close endpoints. If two endpoints are connected, the stream protocol ensures that data written at one endpoint is delivered to the other endpoint. The operation `Create_Endpoint` operation will initiate a TCP socket to listen for a connection request. If a connection is accepted, a function will be attached to the socket and incoming data will be read by that function and delivered to the audio/video producer/consumer object via the stream binder object and stream stub object. The `Connect_flow` operation attempts to make a connection to a socket instantiated by a `Create_Endpoint` operation. If this attempt is successful an identifier for the established flow will be returned (FlowId). This identifier can be used to write on the flow and close the flow by means of the `Write_flow` and `Close_flow` operations.

## 6.4 Case 2: Synchronisation service

An important requirement for multimedia services is the synchronisation within and between related flows. This section studies a synchronisation service relevant for multimedia services that operate in an distributed environment. Synchronisation requirements and policies are described from different stakeholder perspectives (enterprise specification). Synchronisation is then specified in detail using the information, computational and engineering viewpoint languages. Finally, a description of the implementation of synchronisation for a particular multimedia conferencing service in a DPE platform is given. The synchronisation object is proposed as an additional component available for a DPE that needs to deal with time-based flows.

This section is structured as follows: Section 6.4.1 presents the synchronisation issue and provides insight in the various solutions described in the literature regarding flow synchronisation. Section 6.4.2 describes the analysis phase providing the requirements and policies from the different stakeholders involved. Section 6.4.3 and Section 6.4.4 describe the definition and specification phases providing information, computational and engineering specifications of the synchronisation service. The development phase in Section 6.4.6 shows the realisation of the synchronisation function on a DPE platform.

### 6.4.1 Overview of synchronisation

The ability to control and co-ordinate multiple flows, is identified as an important requirement for distributed computing platforms that provide multimedia services [62]. This section focuses on three synchronisation relations of interest for flow exchange in a distributed environment. First, the samples need to arrive in time at the receiver before

display or playout time to maintain the continuity of playback. This is called *intra-stream* synchronisation. Second, *inter-stream* synchronisation is needed to preserve the relationships between the samples (different forms of inter-stream synchronisation exist of which lip-synchronisation is a well known one). Third, all participants in a multimedia service should receive flows at the same time although geographically distributed. This form of synchronisation is called *spatial synchronisation* (See Figure 6.14).

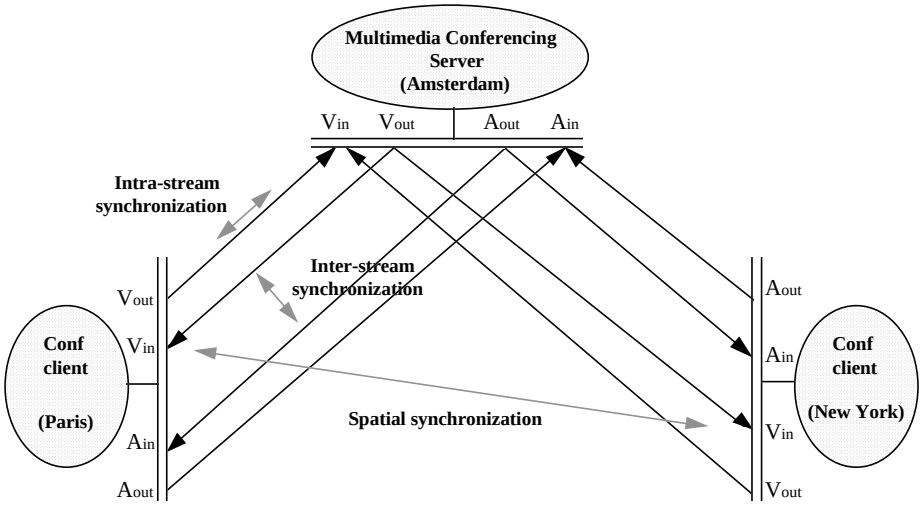



Figure 6.14: Synchronisation in a multimedia conferencing service

This section details the three synchronisation forms, and summarises different techniques that can be used for synchronisation.

## Intra-stream synchronisation

Flows are isochronous in nature. They are sequences of finite sized samples, created at fixed time intervals. Between the flow units, a time relation exists which must be preserved to offer a good quality of presentation. However, several factors can influence the time relation between the flow units:

- Processing and network delay jitter (i.e. the variance in delay);
- Variations in rates of recording and playback;
- Unreliable transmission (i.e. loss) of flow data units.

To deal with network jitter, incoming flow data units can be buffered at the receiver. Starvation caused by processing in the end-nodes can be solved by buffering and careful process scheduling. Jitter can be reduced if the same constant rate is ensured at both the source and sink. However, small variations between source and sink can cause buffer overflow or starvation. A number of solutions have been proposed in the literature to solve these problems. Examples are:

- *Buffer monitoring:* the buffer usage is monitored and beginning overflow or starvation is detected and dealt with [108][109];
- *Feedback technique:* the sink periodically transmits feedback messages to the source, containing the flow unit number that was currently played back [110]. On the basis of these feedback messages, and the bound delay on the network, the source can estimate the actual playback times of flow units at the sink and adjust the flow transmission rate accordingly;
- *Global clock:* synchronisation is achieved by means of a global clock. For example, with the stream synchronisation protocol [113] each flow unit will contain a reference time when it should be played back at the sink. The reference time is calculated at the source of a flow unit using the global time and adding the delay and jitter across the connection.

Losing packets can also result in buffer starvation at the sink. Techniques must be applied to deal with lost or late packets. For intra-stream synchronisation, the maximum jitter caused by transportation and processing must be known. Transportation and processing jitter can be reduced by buffering and scheduling and requires that sufficient buffer space should be available at the sink. A trade-off between buffer space and acceptable delay jitter will need to be made: reducing more jitter will require extra buffer space. Especially with live media, the timeliness of data (i.e. the expiration time of data) should be considered to determine the extent to which delay jitter is acceptable.

Intra-stream synchronisation can be easily achieved by ensuring rate-synchronised clocks (or devices) between the source and the sink. Buffer monitoring is a simple technique to deal with small rate variations of the source and the sink. Feedback and global clock techniques are alternative solutions, but will increase communication (exchange of feedback messages and time reference).

### Inter-stream synchronisation

At the source, a temporal relationship may exist between multiple flows. This relationship must still hold after transportation of these flows through possibly different routes. This is assured when an inter-stream synchronisation mechanism is applied. Solutions proposed are:

- *Multiplexing of streams* (preventive technique): flows are merged into a single flow at the source, and then separated at the sink. During the transport of the flow units, the relationship between these units remains fixed [119];
- *Aggregation in one data structure:* one data structure is defined, which is composed of multiple flow types and their synchronisation relations (e.g. a multimedia document [111]);
- *Global clocks:* this is the same technique as used for intra-stream synchronisation;

- *Synchronisation marker:* at the source, markers are introduced in the flows to indicate sections that should be presented simultaneously [114]. A more advanced technique using logical time stamps is proposed in [115];
- *Synchronisation channel:* a synchronisation co-ordinating node is created. Information about the presentation of the flows, such as playout rates, is periodically transmitted across separate synchronisation channels between the co-ordinating node and the source and sink nodes [116];
- *Feedback technique*: this mechanism is similar as that intra-stream synchronisation although multiple sinks send feedback messages to the source.

Multiplexing is an easy and accurate mechanism to achieve inter-stream synchronisation. Time-based flows are transmitted as one stream which implies that different QoS requirements for individual flows cannot be met (e.g. different jitter bounds for audio and video). synchronisation marker can then provides a good alternative that is easy to implement and provides a similar accuracy.

### Spatial synchronisation

A number of receivers will exist in a distributed multimedia service. With live audio-visual data, it is especially important that all participants receive audio and video data at the same time, to maintain a fair service. Spatial synchronisation mechanisms are used for this purpose. These mechanisms are based on global clocks, synchronisation channel or feedback techniques that are similar to those used for inter-stream synchronisation.

When global clocks are available, mechanisms based on these clocks can achieve the most accurate spatial synchronisation. The stream synchronisation protocol described in [113] is an example of a distributed synchronisation technique whereas the synchronisation channel and feedback technique can be considered as centralised synchronisation techniques.

## 6.4.2 Analysis phase (enterprise specification)

The ODP enterprise language is used to organise the synchronisation requirements and policies in the analysis phase. The ITU SG15 enterprise template is applied to provide a ‘formal’ description as depicted in Table 6.7. The focus is on the service provider, network operator and end-user roles.

| | |
|---|---|
| **COMMUNITY** synchronisation service | |
| **PURPOSE** flow synchronisation | |
| **ROLE** | |
| ***service-provider** | offers the service and is responsible for receiving and broadcasting audio and video flows to the participants. |
| ***network-operator** | provides the end-to-end transport facilities for audio and video flows. |

For audio and video synchronisation strict QoS guarantees on throughput, maximum delay jitter, maximum delay and error rate should be provided.

***end-user** has synchronisation requirements basically determined by the limits of human perception.

**POLICY**

***obligation**

Lip-synchronisation should be in the -20 ms to +40 ms range.
Audio or video jitter should be within the range of 10 ms.
Loss of video frames or audio samples are less than 1% of the total sent.

***exception**

If lip-synchronisation is lost, video frames are dropped to get in line with audio. In case of congestion in the node, video frames will be dropped in favour of audio samples.

**ACTION**

***service provider intra-stream synchronisation**

the service provider will manipulate incoming audio and video flows

**obligation** outgoing flows are within the 10 ms jitter boundary.

***service provider inter-stream synchronisation**

the service provider will manipulate incoming audio and video flows

**obligation** related outgoing audio and video flows are within the -20 and +40 ms range.

***service provider spatial synchronisation**

the service provider will manipulate incoming audio and video flows

**obligation** spatial synchronisation should be in the range of -0.25s to +0.25s.

**obligation** the service provider is responsible that outgoing audio and video flows are played out simultaneously at the receiving users within the 0.25 s boundary.

**exception** users with a transmission delay not tolerable for others, will be prevented from participating, or be allowed to join in a limited way.

***network operator offers compulsory transportation of flows**

**obligation** a transportation service with a deterministic guarantee is provided. This means that in absence of total network failures, the QoS requirements of a client, such as maximum delay, throughput and jitter are met.

**obligation** fixed resource reservation must take place to be able to provide such a service.

**obligation** a user or service provider should not exceed his specification of transport characteristics.

**obligation** If the agreed QoS can not be maintained the service provider or user should be informed.

*** network operator offers statistical reliable transportation of flows**

**permission** to provide transportation service with a certain percentage of QoS violations is provided. Resource reservation is required, but now these resources can be shared with other users of this transportation service.

**obligation** a user or service provider should not exceed his specification of transport characteristics.

**obligation** If the agreed QoS can not be maintained the service provider or user should be informed.

*** network operator offers best effort transportation of flows**

| | |
|---|---|
| **permission** | to change the QoS offered to the client if load conditions change in future. |
| ***compulsory display of audio and video** | |
| **obligation** | synchronisation requirements must be met. Reservation of processing and storage resources and admission control for other applications at the user's node is required. |
| ***statistical reliable display of audio and video** | |
| **permission** | a certain percentage of violations of the synchronisation requirements is allowed. Reservation of processing and storage resources and admission control is also required; |
| ***best effort display of audio and video** | |
| **permission** | possibilities to fulfil the synchronisation requirements are based on current processing and storage activities. If, for example, other applications are executed on the same node, processing resources might not suffice, and the synchronisation requirements will not be met anymore. |

Table 6.7 : Enterprise specification of the synchronisation service using SG15 template

### 6.4.3 Definition phase (information specification)

The definition phase provides a description of the desired effect of the synchronisation service as seen by the stakeholders. Based on requirements identified in the analysis phase, the specification of synchronisation for multimedia services is performed using the ODP information language. For the specification of synchronisation it is necessary to have insight into QoS characteristics of the involved streams, as well as, the relationships between streams. Synchronisation of flows is only possible when the QoS characteristics of the streams and supporting system/transport are known.

Figure 6.15 shows an OMT graphical representation of an *invariant schema* for the synchronisation issue. QoS attributes [97][119], are used to describe the characteristics (e.g. coding, frame rate) of the sources and sinks (objects) used in a multimedia service. Attributes that are relevant for flow synchronisation are identified independently of how they should be implemented.

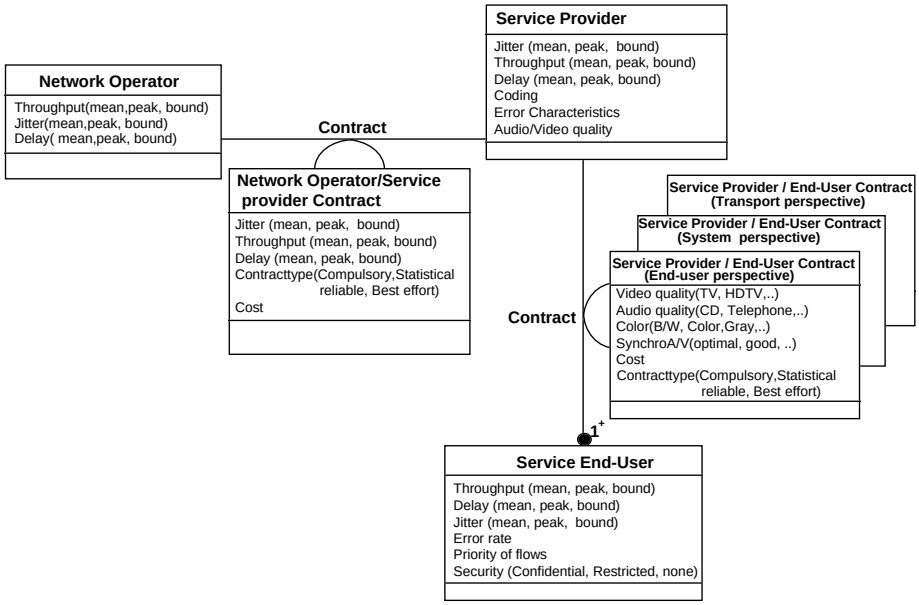



Figure 6.15: Invariant schema of QoS relevant for audio and video synchronisation

For synchronisation it is important to characterise the media types involved by means of QoS attributes. When these attributes are identified, a *contract* can be specified between the stakeholders, which should be respected for the multimedia service to operate properly. A contract provides a specification of the service and 'level of service' agreed between the involved stakeholders. The 'level of service' agreed can be classified as 'compulsory', 'statistical reliable' or 'best effort' as described in Section 5.2. A contract description exists at different levels of abstraction ranging from end-user perspective to detailed system characteristics (see Figure 6.16).

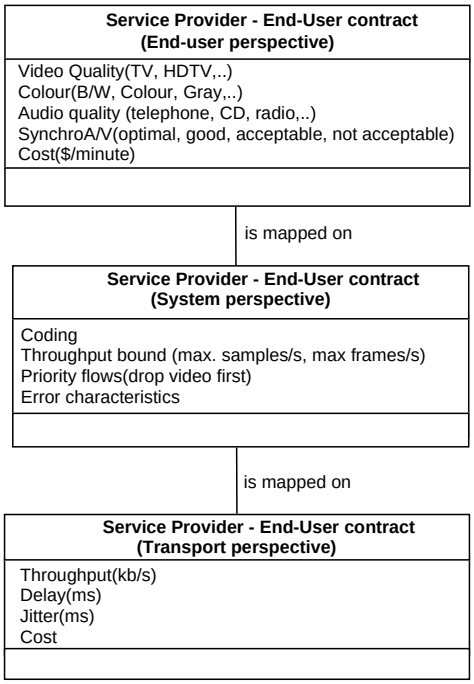



Figure 6.16: Invariant schema for end-user/service provider contract from different perspectives

Figure 6.16 shows an invariant schema for an 'end-user / service provider contract' focusing on QoS attributes relevant for synchronisation. The QoS attributes identified in the different perspectives have a close relationship as illustrated in the following example: a specification by the end-user for television quality, will result in a required frame rate of 25 frames/s from the system perspective. This will in turn require a throughput (depending on the coding technique, picture resolution and colours) of an average of 1.5 Mbit/s at transport perspective.

A *static schema* [16] specifies relevant QoS information at a certain point in time.

Table 6.8 shows an example static schema for a specific service contract. The contract is based on a 'best effort' agreement and applies to all QoS attributes. Values described from the end-user perspective will influence the system attribute values and transport attribute values.

A *dynamic schema* of a contract specifics the possible behaviour subject to the constraints of an invariant schema. For example, in the "service provider - end-user" contract (system perspective), a set of coding techniques are indicated for video flows (i.e. MPEG and H.261). In a 'best effort' agreement this may imply that in case of problems, user and service provider switch from MPEG coding to H.261 coding, which is less demanding for the systems and network.

| QoS attributes | End-user values Video | Audio | System values Video | Audio | Transport values Video | Audio |
|---|---|---|---|---|---|---|
| quality | TV | Phone | | | | |
| costs | $ 10,- per minute | | | | 0.10 $/Mbit | |
| audio and video synchronisation | good | | | | | |
| coding | | | MPEG, H.261 | μ-law | | |
| throughput | | | 25 frames/s | 8000 samples/s | 1.5 Mbit/s | 65 Kbit/s |
| error rate | | | $10^{-9}$ | $10^{-8}$ | | |
| sample size | | | < 8 Kbyte | 1 byte | | |
| synch. accuracy | high quality | | intra-stream {±10 ms} inter-stream {±80 ms} spatial synch {±250 ms} | | | |
| priority flows | | | low | high | | |
| jitter bound | | | | | ± 10 ms | ± 10 ms |
| delay | | | | | <250 ms | <250 ms |

Table 6.8: Example of a static schema for audiovisual flows

### 6.4.4 Specification phase (computational specification)

This section focuses on the operation interface which is used by the synchronisation service to perform management operations.

#### Client management interface

A management interface of a multimedia client must provide operations related to synchronisation. Table 6.9 shows a partial TINA-ODL specification of a client management interface dealing with synchronisation.

```
interface template client_management;

typedef enum  flowtype {audio, video, voice, ... }
typedef enum  Format { MPEG, JPEG, PCM, u-law, ...}
typedef struct  SynchroQoS {
                StreamType  Stream;
                Format      StreamFormat;
                integer     StreamId
                integer     SampleSize, SampleRate, Priority;
                };
typedef QoSParameters = sequence of SynchroQoS;

operations
   QoSnegotiate(in QoSoffer: QoSparameters; out QoSresult: QoSparameters);
   Synchronise (in SpatialDelay, FeedbackInterval: Integer);

behaviour
'This interface enables QoS negotiations with a client. Spatial synchronisation of flows will be performed
according to the parameters supplied by the Synchronise operation'
```

Table 6.9: TINA-ODL specification of client management interface

The `QoSnegotiate` operation enables QoS (re-)negotiations for the exchanged audio-visual flows. As a result, parameters for jitter, sample rate(s), sample size(s) and priority are known to a client, which is then able to apply its local intra-and inter-stream synchronisation mechanism. The `Synchronise` operation is required for spatial synchronisation.

## Server management interface

The multimedia server must support requests from both the clients and network operator to re-negotiate a certain agreed QoS. Table 6.10 shows a partial TINA-ODL specification of a server management interface dealing with synchronisation.

```
interface template server_management;
typedef  client  (...}; /* structure containing information about a multimedia client */

typedef struct SynchronisationInfo{
   integer PlayoutRate, BufferOccupation, CurrentStamp, CurrentTime
};

operations
   QoSrenegotiate(in QoSoffer: QoSparameters; out QoSresult: QoSparameters);
   Synchro_Error (in client: mmClient );
   Synch_Info (in client: mmClient; in SynchInfo: SynchronisationInfo);

behaviour
'An instance of this interface will enable the multimedia server to coordinate synchronisation of several
clients'
```

Table 6.10: TINA-ODL specification of server management interface

The multimedia server supports `Synch_Error` and `Synch_Info` operations to control synchronisation. The first operation enables a client to notify a server about

synchronisation quality problems. Monitoring is required to detect such problems. Spatial synchronisation must be co-ordinated regularly. As a result of a 'best-effort' agreement between the client and service provider, the target playout rates may not be achieved and a client may fall behind. Co-ordination of spatial synchronisation is achieved by synchronisation feedback information contained in the `Synch_Info` operation. This enables the server to perform spatial synchronisation, using either feedback, global clock or synchronisation channel technique. `PlayoutRate` and `BufferOccupation` attributes are required for the synchronisation channel technique. The `CurrentStamp` attribute identifies the last audio unit that is played out and is used by the feedback technique. Together with the `CurrentTime` attribute, indicating the global playout time of that last audio unit, spatial synchronisation can be performed using a global clock.

### Environment contract

The environment contract of computational objects includes statements derived from the requirements and policies described in the analysis phase. The environment contract concept is not developed neither in ODP-RM nor TINA. In Section 5.2.1, Table 5.5 an example is presented for the specification of QoS properties which is also valid for the computational objects involved in synchronisation. QoS information consists of throughput, jitter and delay information. The environment contract for the multimedia server will supply the necessary synchronisation information, such as target delays to each client.

### Behaviour specification

The behaviour specification must include statements on how to maintain synchronisation. The following statements are included informally for a client:

- Related audio and video flows are kept synchronised by speeding up the playout of video samples and by restricting blocking;
- Speeding up or slowing down audio playout is done by skipping audio samples or playing audio samples multiple times;
- Spatial delaying is done by slowing down or speeding up the playout of audio and video samples.

Figure 6.17 shows an SDL diagram [155] illustrating the behaviour of a multimedia client.

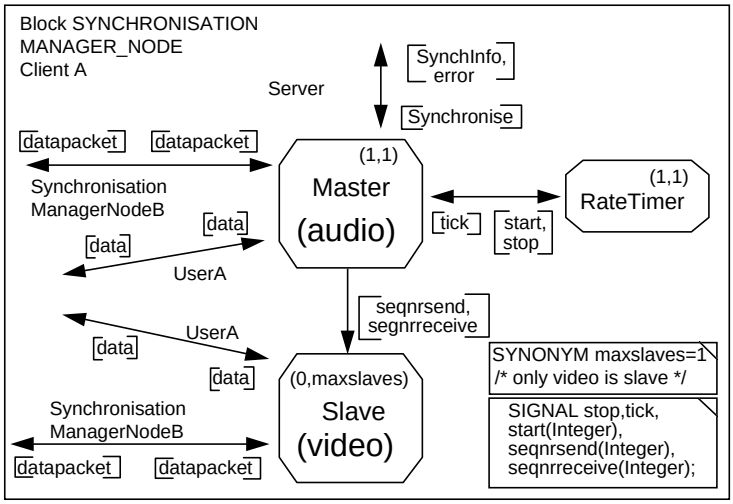



Figure 6.17: SDL description of synchronisation behaviour of a multimedia client

The multimedia server will exhibit the following behaviour related to synchronisation: on successful QoS (re)negotiation it will adapt the values of (spatial) synchronisation parameters. Distribution of these new values to clients is then required. When a client disconnects, the server will adapt the spatial synchronisation. In case of synchronisation errors received from the network operator or multimedia client, the server will initiate QoS renegotiation. An SDL diagram illustrating this behaviour is shown in Figure 6.18.

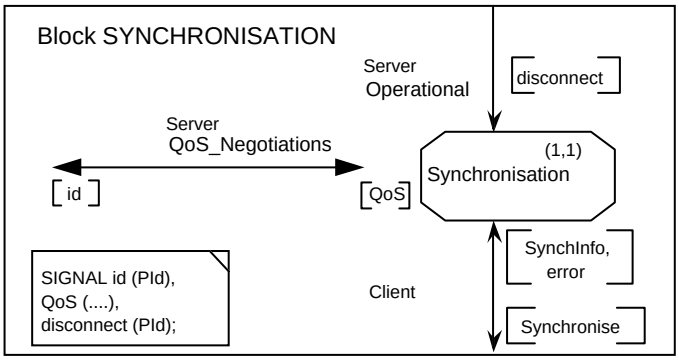



Figure 6.18: SDL description of synchronisation behaviour of a multimedia server

### 6.4.5 Specification phase (engineering specification)

Figure 6.19 shows an engineering configuration of objects that play a role in flow synchronisation. The end-user DPE node provides a set of basic functions such as communication, storage and processing capabilities. It provides the mechanisms to solve distribution transparency and hides the infrastructure from the application designer. Many engineering objects are available in a computer node and required in an

end-user DPE node, and for multimedia applications specific functions should be included (e.g. audio and video device objects).

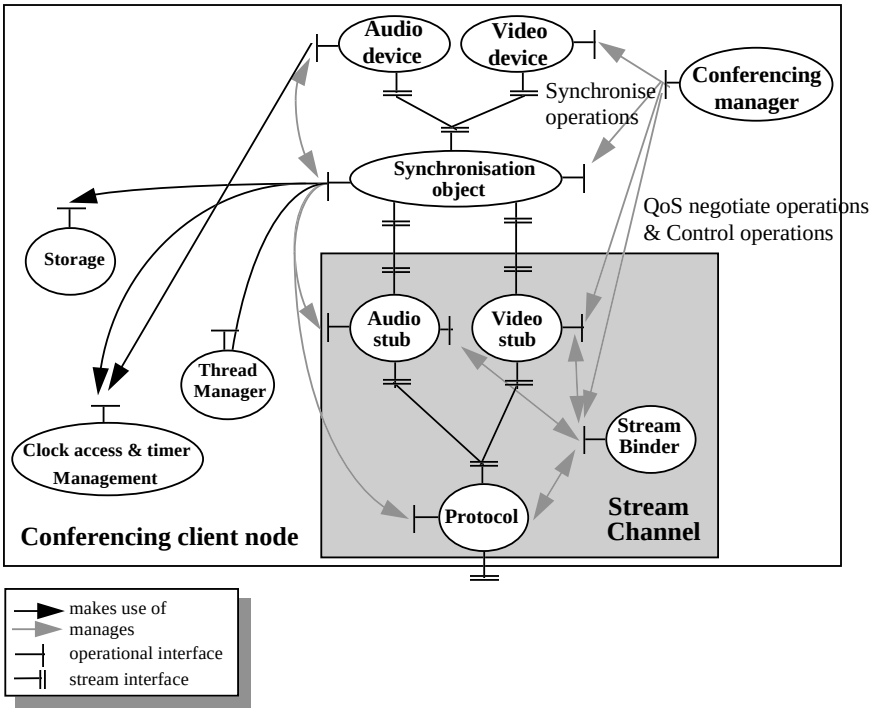



Figure 6.19: Engineering objects involved in synchronisation in the end-user node

The engineering configuration is similar to the configuration as described in Section 5.3.1 and Section 6.3.5. A synchronisation object is added between the original stream interfaces and the channel to fulfil the requirement for synchronisation of incoming flows as stated in the environment contract.

### Node objects related to synchronisation

Several objects in the node are involved in the synchronisation of the time-based flows:

- The *thread manager* allows the use of threads within an object. The thread manager spawns and forks threads. It joins, delays and synchronises threads;
- The *clock access & timer manager* deals with a common distributed notion of time. Time service facilities may provide a common synchronised clock across a number of nodes and delivery of timer interrupts across the distributed system;
- The *storage* object is used to store the samples for further processing. It represents a shared buffer that is used by the other objects to store the samples and by the device objects to playout the samples. The synchronisation object is able to

perform control operations (e.g. drop a sample) on the storage object for flow control;

- The *conferencing manager* object is responsible for the control of the various engineering objects. It performs operations on the device objects such as pausing or resuming a flow. It also controls the channels objects and alters specific QoS parameters when necessary;
- The *audio device* object receives and plays the audio samples. It is a representation of a microphone and speaker;
- The *video device* object is responsible for collecting and presenting video frames. It is a representation of a video camera and screen.

### Synchronisation object

The synchronisation object as depicted in Figure 6.19 is not part of a channel. The use of synchronisation depends on the application. Thus, the application must perform synchronisation according to its needs. Similar ideas can be found in [119], where it is stated that the transport system (i.e. channel) must guarantee bounds on delay and jitter and it is up to the upper layers to realise the synchronisation according to their needs. In [120], it is also stated that the types of operations and corresponding behaviour of the synchronisation object is application-dependent. Also synchronisation outside the channel reduces the jitter in the flows introduced by the node. For example, decompression that occurs in the stub object may increment delay jitter. When performing synchronisation after decompression, the synchronisation accuracy will improve. A synchronisation object can be regarded as a resource available on a node like audio and screen devices. A client can request the node manager for synchronisation of flows.

No explicit control mechanisms are added between the source and sink for intra and inter-stream synchronisation since the mechanisms are applied to live media (e.g. pausing a flow for inter-stream synchronisation is not a very useful operation). Also extensions for multicast transmission are more easily added with a loose coupling between the source and sink [125]. Other advantages are reduction of the complexity of mechanisms and reduction of communication overhead.

*Intra-stream synchronisation:* a node must provide functions to access clocks and timers. These timers can ensure a rate-synchronisation between source and sink. A buffer-monitoring technique is applied to detect starvation or overflow of the buffer. The granularity of the clocks provided by the nucleus must be in the msec range to perform accurate intra-stream synchronisation.

*Inter-stream synchronisation:* the use of threads and the synchronisation of these threads for each audio and video stream results in a reliable implementation. Synchronisation of threads can easily be based on synchronisation markers. The isochrony of digitised audio and video enhances the use of the synchronisation marker

technique. Different QoS requirements for each stream can be met by transporting these flows across different channels. This is a relatively simple and accurate technique: every sample contains a marker holding an identification of the source and a logical time stamp. This type of synchronisation has the advantage that data flows can maintain synchronisation in a network with delayed or lost packets. The time stamps can be used to detect and deal with these packets.

Synchronisation of threads requires real-time scheduling supported by a thread manager and pre-emptive scheduling to ensure delay and jitter bounds [118]. The use of synchronisation markers has an important consequence: the functionality of the stub object needs to be extended since marshalling and unmarshalling of the sequence numbers is required (see stub extension object in 6.4.6).

*Spatial synchronisation:* the design of a multimedia service expects the co-ordination of spatial synchronisation. The multimedia server should support the following spatial synchronisation techniques:

- Feedback technique [110]: the sink periodically transmits feedback messages, containing the stream unit number that was currently played to the source; Due to these feedback messages and the known bound delay on the network, the source can estimate the actual playback times of stream units at the sink and adjust the stream transmission rate;
- Global clock technique [113]: each stream unit contains a reference time that indicates when it should be played out at the sink. This reference time is calculated at the source of a stream unit by taking the global time, to which the delay and jitter across the connection are added;
- Synchronisation channel technique: a synchronisation co-ordinating node is created. Information about the presentation of the flows, such as playout rates, is periodically transmitted across separate synchronisation channels between co-ordinating node and source and sink nodes.

If a global clock exists amongst multimedia clients, spatial synchronisation will be co-ordinated using that clock. Otherwise, the feedback technique will be applied, unless delays for remote operations are not known precisely. In this case, the synchronisation channel technique will be used with the evaluation of playout rates of the multimedia clients.

### 6.4.6 Development phase (technology specification)

The development stage in the TINA life cycle model consists of the development of hardware and software for the synchronisation service. ANSAware 4.0 is used for the realisation of the multimedia service. ANSAware 4.0 is extended to support stream interfaces as reported in [121][140]. Based on this extension the synchronisation object is added. For the end-to-end communication the UDP was selected as the protocol for stream communication. The requirement of best effort service for the audio/video

quality in analysis phase implies that a very reliable communication is not required for audio and video data. However, a mechanism was added to detect and replace lost packets (this is necessary for intra-stream synchronisation).

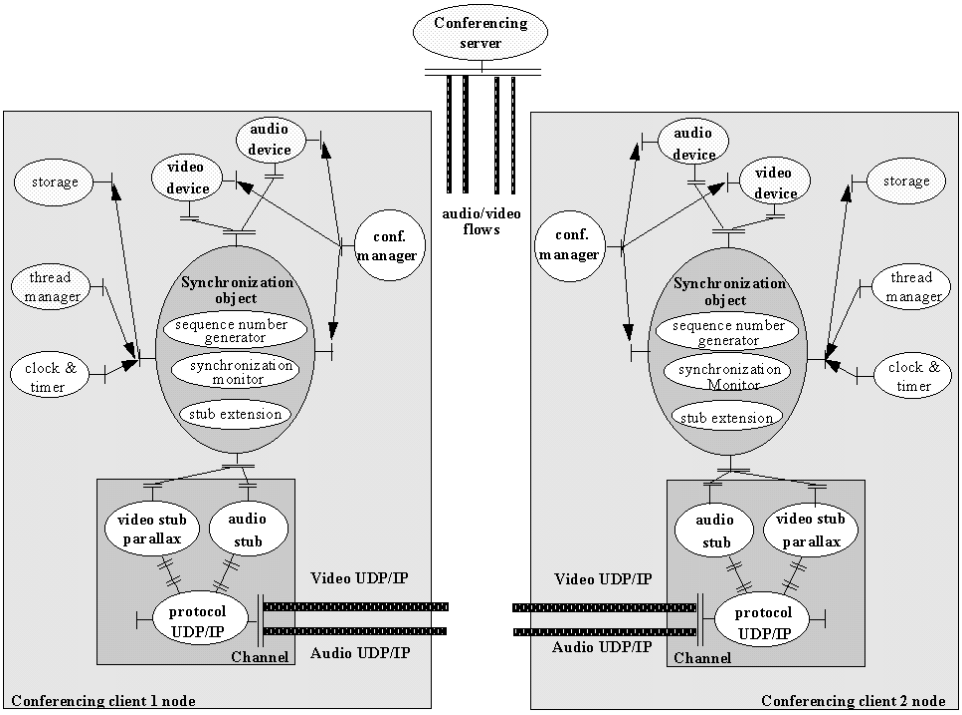



Figure 6.20: Engineering configuration realised on ANSAware 4.0

Figure 6.20 shows the engineering configuration for multimedia clients implemented on ANSAware 4.0. The use of shared buffers is adopted for synchronisation as described in [116][117]. This eliminates the need for synchronous channel/application interaction. At the source, *parallax video board* grabs images from the video camera and stores the samples into the storage object. The *audio grabber* operates in a similar way: the 'grabbing rate' is determined by the synchronisation object, which uses the clock and scheduling functions, and adds a sequence number to the sample. The synchronisation object determines when it is appropriate to offer the samples to the channel object for marshalling and transmission.

At the sink, incoming samples are unmarshalled and stored in the storage object (in this case a shared buffer for audio and video samples). The screen device and audio device retrieve samples from the shared buffers. The retrieval rate is again determined by the synchronisation object. The sequence numbers are used to sort the audio and video samples.

Intra-stream synchronisation is achieved by transmitting samples at the source at a constant rate. At the sink the received data is placed in a buffer (represented as the storage object). The synchronisation object maintains synchronisation by sending the content of these buffers to the devices at a specific rate. The rate is maintained using the internal clock of the SUN Sparc station (the granularity of ANSAware 4.0 clocks were not small enough).

Inter-stream synchronisation is co-ordinated choosing a master sink synchroniser and slave sink synchroniser (audio samples have priority above video samples). The slave follows the master and synchronisation between master and slave is achieved based on the sequence numbers. Since ANSAware 4.0 does not support pre-emptive scheduling, master and slave sequence numbers are kept related by dropping slave stream units first.

Spatial synchronisation is maintained by introducing the spatial delay prescribed by the synchronisation manager of the server. Introducing this delay dynamically can be done by adapting playout rates for a short period. The technique applied by the synchronisation manager of the server must be selected in advance.

## 6.5 Conclusions

The TINA life cycle construction phase has been used as a development method for the specification and implementation of both case studies[21]. Since the TINA construction phase model is very general and does not provide detailed guidelines, it has been refined using the ODP viewpoint languages. This refinement gave enough support to design both case studies. Although this refinement has been a good step forward several issues are identified which need to be improved:

- An important one is the correspondences between the ODP viewpoint specifications. How does the designer know that two (viewpoint) specifications are consistent with each other? Which rules must a designer follow in order to map, for example, an information specification (definition phase) onto a computational specification (specification phase)? These issues still need to be resolved to create a powerful design method suitable to specify distributed telecommunication services;
- A specific problem is the correspondence between engineering specifications and technology implementation. For the case studies this correspondence was rather easily achieved due to the fact that ANSAware is a realisation of the ODP engineering concepts. However, the author expects that DPE platforms such as CORBA, DCE or DCOM will dominate the market. These implementations are not strictly based on ODP engineering concepts and it will be much harder to provide correct correspondences between ODP engineering specifications and the implementation;

21 The case studies have been implemented at KPN Research in ‘94/’95 by two students as part of their master thesis and encompassed nearly 18 months effort.

- In the analysis phase the enterprise viewpoint been refined using the SG15 enterprise language. This language is rather simple and several weak points were identified. The possibility to express a relation between 'actions' is not present. It is desirable to have the possibility to express which actions are related in order to determine the dependency amongst relations. The SG15 enterprise language only supports the semantics to allow the designer to list the actions. Another omission is a clear difference between action and activity. Also the means to describe the relations between actions is not developed and the question arises how the SG15 enterprise language is to be used without resulting in an 'algorithm', which is outside the scope of the enterprise language. A natural language such as English provides the same benefits as the SG15 enterprise language except for the (little) guidance offered by the SG15 template.

### Design method

Neither TINA nor ODP-RM have the intention to develop a detailed design method for distributed telecommunication services. Only general guidelines are provided that should be (hopefully) enough to develop interoperable products. A precise design method for the development of distributed services is often company specific and regarded as a strategic means to create services faster and more efficient than competitors. This means that it can not be expected that companies want to standardise their design method due to strategical reasons. More general design methods such as those developed by Booch, Rumbaugh or Jacobson are more appropriate for standardisation due to their company independent character. To bridge the gap between the very general TINA/ODP-RM guidelines and the company specific design method, the author believes that a possible solution can be found in the Unified Modelling Language (UML). UML includes a design method and could serve as a further refinement of the ODP viewpoint languages (see Figure 3.7 for a relationship between TINA, ODP and UML). Although no experience is obtained yet with UML, the author expects that UML compliant CASE tools will be able to check consistency between the UML diagrams (and thus indirectly between ODP viewpoint specifications). It should be noted that when using UML, the designer might use different concepts than the ODP specific ones.

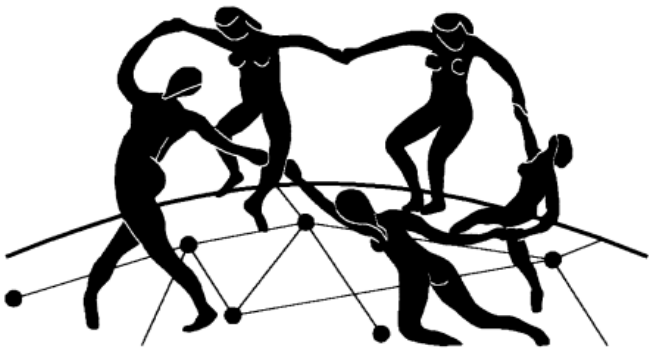
### Multimedia conferencing Service

Regarding the multimedia conferencing service the emphasis was on the specification of the conferencing client interfaces. The author believes that for successful interoperability between conferencing services developed by different companies, standardisation of these external interfaces is of prime importance. The use of an interface definition language, which is implementation independent is promoted. The most appropriate way to standardise distributed telecommunications services is the standardisation of the interfaces supported by a distributed service. It increases the usage of services by clients since the clients do not depend on one specific

implementation and the server can be used by a wider community. Additionally, it creates the possibility to compare different implementations of services.

### Synchronisation

With respect to the synchronisation case study the author supports the view that a synchronisation object should become a basic component of a DPE node that supports real-time multimedia applications. This will enable designing applications from a higher abstraction level. Similar ideas are found in the Microsoft MiTV platform where a synchronisation service is available (see also Section 4.5). A multimedia service can achieve synchronised flows by requesting this engineering object to perform this task. The application design will abstract away from the issue how synchronisation is achieved. Additional engineering objects need to be available in the node, such as a thread manager and a clock access & timer manager.



# 7 Conclusions

*This thesis focused on multimedia services in open distributed telecommunication environments. This issue was approached from a hybrid standardisation philosophy using the concepts and principles of ODP-RM as a basis and complemented with de-facto standards. This chapter discusses the author's view on the use of de-facto and de-jure standards for the specification of distributed multimedia services. It discusses open issues and perspectives in the domain studied.*

## 7.1 Introduction

This thesis explained and discussed the concepts, rules and languages as defined by the de-jure standard ODP-RM. Insight was provided about the use and usefulness of ODP-RM for open distributed service specification. Also, an overview was provided of several industrial bodies involved in de-facto standardisation of distributed processing environments (DPEs). This overview gave insight in various (often different) solutions for the DPE. Special attention was paid to multimedia concepts present in these DPEs. The thesis then focused on the ODP computational and engineering viewpoint and ODP concepts for multimedia were detailed. It showed how the various multimedia concepts described in de-facto standards could be integrated into the ODP-RM framework. Further, ODP-RM was used for the design of distributed multimedia services. Concepts and rules as defined by TINA and OMT were used to model the case studies. These two case studies showed the use of the concepts developed in this thesis.

This chapter gives an overview of the issues and perspective in the research domain covered by this thesis.

## 7.2 ODP-RM: the standard for distributed processing?

The author has contributed to the creation of the ODP-RM standard since 1991 and attended most of the international ISO/ITU SC21/WG7 standardisation meetings. Having applied the standard to several case studies, the author obtained practical experience with the use of ODP-RM. Based on this experience, he believes that ODP-RM provides several benefits but he does not regard ODP as THE standard that is closely being followed by the industry. A close examination of the work in the industry shows that ODP principles and rules are often used loosely and sometimes even conflict with the ODP concepts and rules.

Nonetheless, ODP-RM is a good instrument to understand distributed processing and it is useful for people to position their work. The advantage of ODP is that it provides a framework of different abstractions which is very useful in dealing with the complexity of distributed systems. Using the same abstraction principles (i.e. the five ODP viewpoints) designers of distributed services can easily communicate ideas. Also, the standardised terminology of ODP-RM contributes to this point. On the other hand, the author regards the very generic description of the ODP viewpoint languages as a weak point. It allows too much interpretation freedom for people that use the ODP viewpoint languages. This results in incompatible solutions even when the ODP concepts are used correctly. The enterprise, information and technology viewpoints particularly lack necessary details. To solve this problem, the author suggests that ISO/ITU-T applies a fast track procedure for the enterprise and information viewpoint languages and incorporates existing standards to be a ‘practical’ implementation of an ODP viewpoint language. For multimedia specifications in the information viewpoint, the author suggests to use UML or OMT specification techniques as shown in the case studies. For the enterprise language, a natural language can be used to specify the multimedia requirements for an application. With respect to the technology language, it will be very hard (and even undesirable) to fast track a particular standard due to the fast progress of technology that will outdate such a standard quickly.

## 7.3 ODP-RM: the overall framework for de-facto and de-jure standards?

The author noticed that the industry moves away from de-jure standards in favour of de-facto standards for distributed processing. This is understandable since technology evolves very fast and the industry is not willing to wait for the output of de-jure standards that sometimes take years to become stable. Due to tough competition in DPE products (e.g. Microsoft DCOM, OMG CORBA, OSF-DCE), the industry bypasses the ODP standard and provides its own solution. Although ODP is sometimes acknowledged to be useful (e.g. by OMG), the reality shows that companies mainly invest in de-facto standards from which it is expected to provide results. However, no

single winner is identified yet and many companies participate in several standardisation groups and consortia to be sure not to miss the winning solution. However, the author does not believe in a single winner and therefore promotes in Chapter 3 a hybrid solution of de-jure and de-facto standards for the open distributed processing environment.

In this scenario, ODP-RM forms an appropriate basis for distributed processing standards. De-jure standardisation organisations (ISO/IEC and ITU-T) can build reference frameworks that establish coherent perspectives for standard development. De-facto standardisation efforts can mainly provide technical components that often lack overall consistency and have a limited lifetime and evolution. Hybrid standardisation of distributed processing environments offers a win-win situation in which de-jure standards provide a long-term framework and de-facto standards supply up-to-date specifications of components in a reasonable time. Distributed multimedia systems based on a long term (implementation and technology neutral) architecture (i.e. ODP-RM) can evolve and preserve the investments made in distributed systems. In time, components can be replaced with newer versions without major modifications to the system architecture.

## 7.4 Refinement of ODP multimedia concepts using de-facto and de-jure standards

Chapter 3 and 4 investigated the main players in the field of distributed processing and special attention is paid to multimedia concepts to construct distributed applications. From this study it is seen that only TINA and ODP-RM provide multimedia concepts that are independent from any particular implementation. On the contrary, IMA, DAVIC and MiTV define multimedia concepts specifically for their implementation environment. These multimedia concepts differ considerably. Based on this inventory, the author concludes that only at a higher abstraction level (i.e. computational and engineering) standardisation is useful. The terminology and concepts as defined by ODP-RM provide a good starting point for the alignment of these de-facto standards. Each consortium should then do a particular mapping of these implementation independent specifications onto specific (multimedia) standards and technology independently.

Chapter 5 demonstrated the use of de-facto results for the specification of an ODP conform multimedia binding object. The descriptions of IMA, OMG and TINA were used for the specification of the control interfaces of a multimedia binding object. Further standardisation is needed to align these descriptions to a consistent specification of a general multimedia binding object. The author promotes the use of a specification language such as TINA-ODL to obtain implementation independent descriptions.

The QoS framework defined by ISO/IEC was used as input for the specification of a general structure for the ODP environment contract which is used to specify the QoS properties in the computational viewpoint. The proposed structure for the environment

contract is very flexible. It allows various ways to express the QoS constraints on interfaces and objects. This flexibility is necessary since the environment contract has to accommodate many different types of objects, which operate in different environments and thus require different QoS specifications. Research is needed to complete the computational QoS specifications for multimedia applications and in particular their relationship to (network and end-system) resources should be studied. Additionally, the environment contract needs to be extended with non-multimedia specific QoS parameters such as security and reliability.

TINA-ODL as defined by the TINA consortium was used to express the syntax for stream signatures. The author introduced a different syntax for stream signatures compared to operation signatures to clearly express the difference in the interaction paradigm between stream interfaces and operation interfaces. The proposed syntax for stream interfaces allows for the complete specification of time-based flows specifying their media characteristics including QoS descriptions.

When the author started to work on ODP multimedia concepts such as stream interface, binding object and environment contract they were unclear and most people did not know how to use them in the specification process. As of 1996, it is a hot issue exemplified by OMG that requests the industry to provide solutions how to deal with time-based flows in a CORBA environment. The author contributed to make these concepts more understandable by providing input to the ODP-RM standard (e.g. Section 12 of the ODP-RM Part 1, [14]) and by contributing to TINA-C deliverables [66], several articles [11][51][123], and OMG through a white paper [88].

## 7.5 ODP-RM and the telecommunications environment

The Telecommunication Information Networking Architecture (TINA) is a telecommunications based solution for the development of distributed services and DPEs. The author participated in the TINA core team in the USA for one year and was responsible for the specification of a general DPE [53] and introduced multimedia concepts in the TINA deliverables [66]. The results are based on ODP-RM and many topics described in this thesis can also be found in TINA deliverables.

For multimedia, TINA adopts the ODP-RM concepts and refines them towards their specific solution for setting up connections. TINA is based on a hierarchical approach for connection establishment. The author questions, however, to which extent this approach is going to be accepted by the industry. The fact that a third party (service provider) is responsible for the connection set up and not the end-user themselves might cause problems since connection establishment is normally initiated by the end-user without the explicit involvement of a third party. Prototypes [65] implementing (part of) the TINA architecture also show that connection establishment is made using signalling protocols where the end-user initiates connection set up. This implies that the TINA connection model is appropriate for services where a third-party (service provider) is needed that requests more complex connectivity but the connection model is not

optimal for connection set up without the use of a third party service provider (e.g. in LAN environments).

TINA mainly focused on the generic parts of a distributed service. Other initiatives such as DAVIC and IMA have developed more detailed specifications for service control of multimedia services. The author believes that additional work is needed to investigate how the TINA generic service control specifications and the service specific control developed in other consortia relate and hopefully complement each other. From TINA's perspective, this requires that closer alignment with these consortia should be sought and where possible they should fast-track these service specific specifications as part of the TINA architecture service specification.

TINA defines a reference architecture for open telecommunications systems making use of distributed computing and OO technology to achieve interoperability in a multi-vendor environment and reusing software and applications. The author regards the TINA initiative as a means to push IT issues such as object orientation and distributed processing environments in the telecommunications industry. It is the author's opinion that TINA shows a direction on where the telecommunications industry should go but it can not be expected that TINA will provide the detailed specifications that will be standardised by telecommunication equipment providers.

## 7.6 Issues for further research

**Design method for distributed multimedia services**

An important objective of this thesis was to investigate separation principles useful for the analysis and design of distributed multimedia applications. In this thesis, the author advocates the use of a combination of separation principles (i.e. TINA life cycle, ODP viewpoint languages and UML) as discussed in Chapter 3. It covers the whole life cycle of a service and in addition it provides the hooks on how to analyse and design a service. The initial results obtained by applying it to the case studies has given the author confidence that this design method is a useful means to describe complex distributed services.

However, the author identified the need for CASE tools that support the designer. CASE tools based on the Unified Modelling Language (UML) are, in the authors view, a good candidate to develop services for DPEs. Due to the importance of the de-facto standard UML (which might become an ODP standard in the future), the author described in Chapter 3 a possible integration of UML in the ODP-RM framework. Research is needed to obtain practical experience with the use of UML concepts and their exact positioning along the ODP viewpoints. The author believes that a combination of the three separation principles (i.e. TINA life cycle, ODP viewpoints and UML) will provide useful guidelines for a designer to develop distributed services.

### Tools for translating QoS specifications

The focus has been in this thesis on the design of services but an issue for further research is the refinement of the deployment, operation and withdrawal of services. An interesting issue for further research is to extend the design method with tools that support the development of multimedia services incorporating the specific QoS requirements. In particular, tools that assist in translating QoS specifications to system characteristics would be beneficial. An important issue that needs further research is the automatic translation of a computational multimedia specification onto a corresponding engineering one. Research is needed to derive rules/mechanisms on how QoS specifications in the computational viewpoint can be automatically mapped onto engineering QoS characteristics of the systems and networks used to implement the computational specification. Ideally, a compiler would exist, which takes into account the computational specification with a QoS description and automatically creates the corresponding engineering configuration which complies to the computational QoS specification. The author does not know any compiler that includes the QoS specifications. This implies that the engineering specification needs to be checked manually and probably, be extended with additional objects (e.g. QoS managers) and mechanisms (e.g. QoS negotiation protocols, QoS monitors) to guarantee the computational QoS description.

### Multimedia in DPEs

Multimedia in distributed processing environments is certainly going to happen. Driving forces like Internet, CORBA and Microsoft offer a variety of tools that allow the creation of flashy multimedia applications and colourful multimedia web pages. Although emerging, the QoS guarantees and real-time characteristics are still rather limited. The need for sufficient bandwidth and QoS guarantees offered to services will make it possible that real-time multimedia is successful in distributed processing environments.

# 8 References


[1] C. Chapel, C. Guilfoyle, J. Jeffcoate, *'Middleware: making the right decisions'*, Technology Appraisals, ISBN 1-871802-33-4, 1995.

[2] R.Orfali, D. Harkey, J. Edwards, *'The Essential Distributed Objects Survival Guide'*, ISBN 0-471-12993-3, John Wiley&Sons, Inc., 1995.

[3] P. Teale, *'Middleware--or Muddleware?'*, Version 5, IBM Intranet and client/server, IBM UK Ltd, www.csc.ibm.com/advisor/library/ap251e.htm, January 1994.

[4] R.Rock-Evans, *'Middleware: the key to distributed Computing'*, OVUM Reports, ISBN 1-898972-45-1, 1995.

[5] P. Manchester, *'Standards bodies and middleware'*, Middlewarespectra, pp. 40-45, Spectrum Reports Limited, February 1995.

[6] P. Leydekkers, L.J.M. Nieuwenhuis, *'Telecommunications Information Network Architecture (TINA) Consortium'*, Handboek Telematica, Samson Bedrijfsinformatie BV, Kluwer, 1995.

[7] A. Reinhardt, *'Building the Data Highway'*, Byte, pp. 46-74, March 1994.

[8] K. Anstötz, *ATM - Success or Failure? Economic remarks on the introduction of Broadband Technology in Europe*, International Journal of Digital and Analog Communications Systems, Vol.6, pp. 161-165, 1993.

[9] M.J. Hoogeveen, F.C.I. van den Eijnden, *'Principles of a public multimedia information service'*, Multimedia/Hypermedia in Open Distributed Environments, Proceedings of the Eurograhis Symposium in Graz, Austria, pp. 301-315, Springer-verlag, June 1994.

[10] KPN Nieuwsblad, *'Diensten belangrijker dan infrastructuur'*, Jaargang 07 – 20, Nummer 158, June 1996.

[11] M. Jørgensen, P. Leydekkers, M. Mampaey, H. Yang *'Support for Distributed Multimedia Services in the TINA Architecture'*, Proceedings of the third Internal Symposium on Autonomous Decentralized Systems (ISADS97), Berlin, April 1997.

[12] L.J.M. Nieuwenhuis, *'Overbruggen van afstanden'*, Inaugurele rede, Rijksuniversiteit Groningen, 1995.

[13] L.J.N. Franken, *'Quality of Service Management: a Model-Based Approach'*, CTIT Ph.D.-thesis series No. 96-10, University of Twente, ISBN 90-72125-56-8, 1996.

[14] ITU/ISO, Open Distributed Processing - Reference Model, *'Part 1: Overview'* International Standard 10746-1, ITU-T Recommendation X.901, 1996.

[15] ITU/ISO, Open Distributed Processing - Reference Model, *'Part 2: Foundations'*, International Standard 10746-2, ITU-T Recommendation X.902, 1995.

[16] ITU/ISO, Open Distributed Processing - Reference Model, *'Part3: Architecture'* International Standard 10746-3, ITU-T Recommendation X.903, 1995.

[17] Open Distributed Processing - Reference Model, *'Part 4: Architectural Semantics'* Draft International Standard 10746-4, ITU-T Recommendation X.904, 1995.

[18] K. Farooqui, L. Logrippo, J. De Meer, *'The ISO Reference Model for Open Distributed Processing: an introduction'*, Computer Networks and ISDN Systems 27, pp. 1215-1229, 1995.

[19] TINA coreteam, *'Connection Management Architecture'*, Document No. TB_JJB.005_1.3_94, TINA-C, January 1995.

[20] OMG, *'The Common Object Request Broker: Architecture and Specification'*, Revision 2.0, OMG Technical Document PTC/96-03-04, July 1995.

[21] V. Gay, P. Leydekkers, R. Huis in 't Veld, *'Specification of Multiparty Audio and Video Interaction Based on the Reference Model of Open Distributed Processing'*, Computer Networks and ISDN Systems – pp. 1247-1262 Vol. 27 Number 8, July 1995.

[22] F. Boussinot, G. Doumec, J.B. Stefani, *'Reactive objects'*, CNET Internal publication, 1995.

[23] ISO/ITU, *'A note on issues on QoS and ODP'*, ISO/IEC JTC 1/SC21, November 1995.

[24] P. Leydekkers, V. Gay, *'ODP View on Quality of Service for Open Distributed Multimedia Environments'*, Proceedings of the 4th International IFIP Workshop on Quality of Service (IWQoS 96), J. de Meer and A. Vogel (Eds.), Paris, March 1996.

[25] OMG, *'CORBA 2.0/Interoperability - Universal Networked Objects'*, OMG Document number 95.3.10, March 1995.

[26] Y. Hoffner and B. Crawford, *'Federation and Interoperability'*, APM.1514.01, ANSA Phase III, October 1995.

[27] OMG, *'Common Business Objects and Business Object Facility'*, OMG TC Document CF/96-01-04, January 1996.

[28] EURESCOM Project 517, Foresight Study on Distributed Object-oriented Computing, *'Distributed Objects in Telecommunications: A Manager's Guide'*, March 1996.

[29] EURESCOM Project 517, Foresight Study on Distributed Object-oriented Computing, *'Technology Life cycle Model'*, April 1996.

[30] EURESCOM Project 517, Foresight Study on Distributed Object-oriented Computing, *'Maturity Rating of Distributed Object-oriented Technologies'*, March 1996.

[31] C. Chapel, C. Guilfoyle, J. Jeffcoate, *'Middleware: making the right decisions'*, Technology Appraisals, ISBN 1-871802-33-4, 1995.

[32] Multimedia System Services, IMA Recommended Practice, *'Part 1: Functional Specification'*, Second Draft, September 1994.

[33] Multimedia System Services, IMA Recommended Practice, *'Part 2: Multimedia devices and formats'*, Second Draft, September 1994.

[34] Multimedia System Services, IMA Recommended Practice, *'Part 3: Transport and media stream protocol specifications'*, First Draft, September 1994.

[35] A. Cramer, M.farber, B. McKellar, R. Steinmetz, *'Experiences with the Heidelberg Multimedia Communication System'*, High Performance Networking 92, pp. D4:1-19, IFIP, 1992.

[36] H. Rubin, N. Natarajan, *'A Distributed Software Architecture for Telecommunication Networks'*, IEEE Network, January/February 1994.

[37] Digital Audio-Visual Council, DAVIC 1.0 specification, *'Part 1: Description of DAVIC functionalities'*, Technical Report, January 1996.

[38] Digital Audio-Visual Council, DAVIC 1.0 specification, *'Part 2: System Reference Models and scenarios'*, Technical Report, January 1996.

[39] Digital Audio-Visual Council, DAVIC 1.0 specification, *'Part 3: Service Provider System Architecture and Interfaces'*, Technical Report, January 1996.

[40] Digital Audio-Visual Council, DAVIC 1.0 specification, *'Part 4: Delivery System Architecture and Interfaces'*, Technical Report, January 1996.

[41] Digital Audio-Visual Council, DAVIC 1.0 specification, *'Part 5: Service Consumer System Architecture and High Level API'*, Technical Specification, January 1996.

[42] Digital Audio-Visual Council, DAVIC 1.0 specification, *'Part 7: High and Mid-layer Protocols'*, Technical Specification, January 1996.

[43] Digital Audio-Visual Council, DAVIC 1.0 specification, *'Part 8: Lower-Layer Protocols and Physical Interfaces'*, Technical Specification, January 1996.

[44] Digital Audio-Visual Council, DAVIC 1.0 specification, *'Part 9: Information Representation'*, Technical Specification, January 1996.

[45] X/Open Consortium Specification *SPIRIT Platform* Blueprint, SPIRIT Issue 3.0, 1995.

[46] A. Herbert, *'An ANSA Overview'*, IEEE Network, pp. 18-23, January/February 1994.

[47] M. Milliking, *'DCE: Building the Distributed Future'*, BYTE magazine, June 1994.

[48] M. Sherman, *'DCE:dead in the water or rising with the tide?'*, Middlewarespectra, May 1995.

[49] M. Hubley, *'Open Software Foundation'*, DATAPRO IU08-120-101, Datapro Information Services Group, December 1992.

[50] TINA coreteam, *'TINA-C Glossary of Terms'*, Version 1.0, TINA-C Overall Deliverable, April 1996.

[51] A.T. van Halteren, P. Leydekkers, H.B. Korte, *'Specification and Realisation of Stream interfaces for the TINA-DPE'*, Proceedings of TINA'95 Conference, Melbourne Australia, February 1995.

[52] Bellcore, *'The Operations Systems Computing Architecture: Semantic Integrity of the Totality of Corporate Data'*, Proceedings of the First International Conference on System Integration; TH0309-5/90/0000/0482$01.00 1990 IEEE, April 1990.

[53] Tina coreteam, *'TINA Distributed Processing Environment (TINA-DPE)'*, Document No. TB_PL001_1.0_95, TINA-C, July 1995.

[54] *'Quality of Service - Methods and Mechanism'*, Working draft #3 ISO/IEC JTC1/SC21/N 9681, November 1995.

[55] *'ODP interface References and Binding'*, Working draft, ISO/IEC JTC 1/SC21, SC21/WG7 Interim Meeting, Kobe, Japan, November 1995.

[56] Tina coreteam, *'Engineering Modelling Concepts (DPE Kernel Specification)'*, Document No. TR_KMK.001_1.1_94, November 1994.

[57] A. Vogel, B. Kerherve, G. Bochmann, J. Gecsei, *'Distributed Multimedia Applications and Quality of Service - A Survey-'*, IEEE Multimedia, Vol.2, No2, pp. 10-19, 1995.

[58] K. Nahrstedt, R. Steinmetz, *'Resource Management in Multimedia Networked Systems'*, IEEE Computer, Special issue on Multimedia, May 1995.

[59] L.J. Vermeulen et. al, *'Een aanzet tot een visie op multimedia'*, KPN Research internal document, 1993.

[60] Tina coreteam, '*Computational Modelling Concepts'*, Document No. TB_A2.HC.012_3.0_95, TINA-C, August 1995.

[61] OMG, *'Multiple interfaces and Composition'*, RFP, Document orb/96-01-04, January 1996.

[62] OMG, *'Control and Management of A/V Streams'*, RFP, Document:telecom/96-08-01, August 1996.

[63] Tina coreteam, *'Service Architecture'*, Document No. TB_MDC.012_2.0_94, TINA-C, March 1995.

[64] T.G. Eijkman, V.W. Stinessen, *'OO-methoden en –tools en drie-lagen-modellering'*, KPN Research, R&D-RA-94-1004, December 1994.

[65] H.B. Korte, M.R. Schenk, *'Request for information to vendors regarding platforms for the service pilot on TINA'*, KPN Research, R&D-RA-96-160, 1996.

[66] Tina coreteam,'*Connection/Session Graph, Stream Interfaces and Channel Model'*, Document No. TR_PL.001_1.1_95, TINA-C, February 1996.

[67] Tina coreteam, *'Information Modelling Concepts'*, Document No. TB_EAC.001_3.0_94, TINA-C, December 1994.

[68] Tina coreteam, *'Service Session Control Specifications'*, Document No. TR_MJM.005_2.1_95, TINA-C, February 1996.

[69] Tina coreteam , *'TINA Object Definition Language (TINA-ODL) manual'*, Document No. TR_NM.002_1.3_95, TINA-C, June 1995.

[70] Tina coreteam, *'Management architecture'*, TINA-C, December 1994.

[71] P. Leydekkers, L.J.M. Nieuwenhuis, *'Telecommunications Information Network Architecture (TINA) Consortium'*, Telecommagazine Nr 8., October 1996.

[72] T. Handegard, P. Leydekkers, *'TINA Distributed Processing Environment'*, Global Communications, 1996.

[73] M. Jacobs, P. Leydekkers, *'Specification of Synchronisation in Multimedia Conferencing Services using the TINA Life cycle Model'*, Distributed Systems Engineering Journal 3, The British Computer Society, The Institution of Electrical Engineers and IOP Publishing Ltd, pp. 185-196, November 1996.

[74] L. Hazard, F. Horn, J.B. Stefani, *'Towards the Integration of Real-time and QoS handling in ANSA architecture'*, NT/PAA/TSA/TLR/3498 - 93 - Restricted distribution, 1993.

[75] I.J.P. Kwaaitaal, P. Leydekkers, L.J. Teunissen, *'The Good, the bad and the ugly about Multimedia Conferencing Services'*, Second International Symposium on Autonomous Decentralized Systems, ISADS95, IEEE Computer Society Press, Phoenix USA, April 1995.

[76] XTP Forum, *'Xpress Transport Protocol Specification - XTP Revision 4.0'*; XTP Forum, Santa Barbara, California, USA, March 1995.

[77] MPEG, *'Coding of Moving Pictures and Associated Audio'*, ISO/IEC 11172, International Standard, November 1995.

[78] ISO-13522(5): *'MHEG: Information technology - Coding of Multimedia and Hypermedia Information , Part 5: support for base-level interactive applications'*, International Standard, July 96.

[79] T. Meyer-Boudnik, W. Effelsberg, *'MHEG explained'*, IEEE Multimedia, pp. 26-38, 1995.

[80] B. Kervella, *'MHEGAM: a Complete Multimedia Messaging Based on X.400/MIME and MHEG'*, Ph.D. document, Paris VI University, MASI report Th95.01, June 1995.

[81] ITU/ISO, *'Quality of Service - Framework'*, ISO/IEC CD 13236.2, October 95.

[82] ITU/ISO, *'Information Technology-Quality of Service-Guide to Methods and Mechanisms'*, Proposed Draft Technical Report, ISO/IEC JTC1/SC21/N1022, November 1995.

[83] V. Gay, P. Leydekkers, *'ODP-RM for multimedia'*, IEEE multimedia - standards, pp. 68-73, January-March issue, 1997.

[84] J. Gosling, H. McGilton, *'The Java(tm) Language Environment: A White Paper'*, http://www.student.nada.kth.se/javadoc/whitepaper/java-whitepaper-1.html, 1996.

[85] The ATM forum, *'ATM UNI Version 3.1'*, Spec. Part I, Appendix A.

[86] M. Zweiacker (editor), *'Network Resource Model'*, Eurescom Project P103, technical report 7, December 1994.

[87] K. Raymond, *'Streams and QoS: a White Paper'*, Contribution to OMG Telecommunications Special Interest Group (TELSIG), December 1995,

[88] P. Leydekkers, M. Jørgensen, *'Stream Interfaces'*, TINA-C, Contribution to OMG Telecommunications Special Interest Group (TELSIG), December 1995.

[89] R. Steinmetz, *'Analysing the Multimedia Operating System'*, pp. 68-84, IEEE Multimedia, Spring 1995.

[90] A. Campbell, G. Coulson, D. Hutchison, *'A Quality of Service Architecture'*, ACM SIGCOMM, Computer Communication Review, June 1994.

[91] R. Gopalakrishna, G. Parulkar, *'Efficient QoS Support in Multimedia Computer Operating Systems'*, Washington university report WUCS_TM_94_04, August 1994.

[92] L. Fedaoui, W. Tawbi, E. Horlait, *'Distributed Multimedia Systems Quality of Service in ODP Framework of Abstraction: A first study'*, Proceedings of ICODP, IFIP, Berlin, September 1993.

[93] P. Leydekkers, V. Gay, L.J.N. Franken, *'A Computational and Engineering View on Open Distributed Real-time MultiMedia Exchange'*, In Lecture Notes in Computer Science - Number 1018 - Springer Verlag
ISBN: 3-540-60647-5, 1995.

[94] J.B. Stefani, *'Computational Aspects of QoS in an object-based, distributed systems architecture'*, 3rd International Workshop on Responsive Computer Systems, Lincoln, NH, USA, September 1993.

[95] J.A. Zinky, D.E. Bakken, R. Schantz, *'Overview of Quality of Service for Distributed Objects'*, IEEE Dual Use Technology Conference, Utica, NY, May 1995.

[96] G. Li and D. Otway, *'ANSA Real-time QoS Extensions'*, APM.1094.00.07, draft - restricted distribution, January 1994.

[97] A. Vogel et. al, *'On QoS Negotiations in Distributed Multimedia Applications'*, technical report TR-891, University of Montreal, 1993.

[98] J.T. Schuilenga, *'QoS Monitoring and Modelling of Continuous Streams in a Distributed Environment'*, M.Sc. thesis KPN Research/ University of Twente, April 1995.

[99] A.Vogel, B. Kerherve, G. Bochmann, J. Gecsei, *'Distributed Multimedia Applications and Quality of Service - A Survey-'*, IEEE Multimedia, Vol. 2, No. 2, pp. 10-19, Summer 1995.

[100] Y. Hoffner, *'Interoperability and Distributed Platform Design'*, APM.1507.01, ANSA Phase III, October 1995.

[101] P.J. Kuhn, *'Quality of Service in ATM Networks'*, Slides, 1995.

[102] L.G. Cuthbert, J.C. Sapanel, *'ATM The Broadband Telecommunications Solution'*, IEE Telecommunications Series 29, ISBN 0 85296 815 9, 1993.

[103] L.J.N. Franken, R.H. Pijpers, B. Haverkort, *'Modelling Aspects of Model Based Dynamic QoS Management by the Performability Manager'*, In Lecture Notes in Computer Science, Springer-Verlag, Volume 794, Proceedings of the 7th International Conference on Computer Performance Evaluation. Modelling Techniques and Tools, Vienna, Austria, 1994

[104] L.J.N. Franken, P. Janssens, B. Haverkort, E.P.M. van Liempd, *'Quality of Service Management in Distributed Systems using Dynamic Routation'*, Proceedings of ICODP`95, Brisbane Australia, February 1995.

[105] D. Ferrari, J. Ramaekers, G. Ventre, *'Client-Network Interactions in Quality of Service Communication Environments'*, In Proceedings 4th IFIP Conference on High Performance Networking, Liege, Belgium, December 1992.

[106] N. Williams, G.S. Blair, *'A distributed multimedia application study'*, Computer Communications, 1992.

[107] P. Leydekkers, V. Gay , *'Open Service Architecture based on ODP : Experience in CASSIOPEIA'*, Proceedings of the Race International Conference on Intelligence in Broadband Services and Networks, Paris, November 1993.

[108] N. Almeida, J. Cabral, A. Alves, *'End-to-End Synchronization in Packet Switched Networks'*, In Proceedings of 2nd international workshop on network and operating system support for digital audio and video conferencing, Heidelberg, Germany, 1991.

[109] L. Dairaine, *'Drift Matching Techniques for Time Signature Conservation Service'*, In Proceedings of BRIS'94 Conference, Hamburg, Germany, June 1994.

[110] S. Ramathan, P. Venkat Rangan, *'Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems'*, The Computer Journal, Vol. 36 No. 1, pp. 19-31, 1993.

[111] G. Pancaccini, F. Stajano, *'Media Composition and Synchronization Aspects in an Interactive Multimedia Authoring Environment'*, Human Aspects in Computing, pp. 337-343, 1991.

[112] R. Steinmetz, *'Synchronization Properties in Multimedia Systems'*, IEEE Journal on selected areas in communications, Vol. 8 No. 3, April 1993.

[113] J. Escobar, D. Deutsch, C. Partridge, '*Flow Synchronization Protocol'*, Report BBN Systems and Technologies Division, 1992.

[114] D. Shepherd, M. Salmony, *'Extending OSI to Support Synchronization Required by Multimedia Applications'*, Computer Communications, Vol. 13 No. 7, pp. 399-406, September 1990.

[115] D. Anderson, G. Homsy, *'A Continuous Media I/O Server and Its Synchronization Mechanism'*, IEEE Computer, Vol. 24 No. 10, pp. 51-57, October 1991.

[116] A. Campbell, G. Coulson, F. Garcia, D. Hutchinson, *'A Continuous Media Transport and Orchestration Service'*, In Proceedings of ACM SIGCOMM'92, August 1992.

[117] D. Ferrari, A. Gupta, M. Moran, B. Wolfinger, *'A Continuous Media Communication Service and Its Implementation'*, In Proceedings GLOBECOM '92, Orlando, Florida, December 1992.

[118] G. Coulson, G.S. Blair, N. Davies, N. Williams,*'Extensions to ANSA for Multimedia Computing'*, In Computer Networks and ISDN Systems, Vol. 1 No. 25, pp. 305-323, 1992.

[119] M. Wassim Tawbi, *'Quality of Service in Multimedia Communication Systems: A study of a Framework and Specification of a negotiation Protocol between Applications'*, PhD thesis, Universite P. et M. Curie, Paris, France, December 1993.

[120] G.S. Blair, G. Coulson, P. Auzimour, L. Hazard, F. Horn, J.B. Stefani,*'An Integrated Platform and Computational Model for Open Distributed Multimedia Applications'*, internal report MPG-92-50, Computing Department, Lancaster University, Bailrigg, Lancaster, UK, 1992.

[121] P. Leydekkers, L.J. Teunissen, *'Synchronization of Multimedia Data Streams in open distributed environments'* Lecture Notes in Computer Science, Network and Operating System Support for Digital Audio and Video, Springer Verlag Heidelberg, November 1991.

[122] P. Leydekkers, P. Leever, A. Mellisse, '*Specification of a Multimedia Conferencing service based on the TINA-C Life cycle model'*, technical report KPN Research, September 1994.

[123] P. Leydekkers, V. Gay, *'Multimedia Conferencing Services in an Open Distributed Environment'*, In Lecture Notes in Computer Science - Number 868 - Springer Verlag - R.Steinmetz editor. Proceedings of the 2nd IEEE International Workshop on Advanced teleservices and High Speed Communication Architectures, pp. 339-352, Heidelberg, September 1994.

[124] ANSA, *'ANSA Reference Manual'*, APM Cambridge Ltd., 1989.

[125] J.C. Pasquale, G.C. Polyzos, E. Anderson, V.P. Kompella, *'The Multimedia Multicast Channel'*, In Proceedings of 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, California, November 1992.

[126] T. Ohmori, K. Maeno, S. Sakata, H. Fukuoka, K. Watabe, '*Distributed Cooperative Control for Sharing Applications Based on Multiparty and Multimedia Desktop Conferencing System: MERMAID'*, C&C Systems Research Laboratories, pp. 539-546, 1992.

[127] T. Hiroya, A. Tomohiko, M. Shigeki, S. Kazunori, *'Personal Multimedia-Multipoint Teleconferencing System'*, IEEE INFOCOM, 1991.

[128] A.J. MacCartney, G.S. Blair, *'Flexible trading in distributed multimedia systems'*, Computer Networks and ISDN Systems Vol. 25, pp. 145-157, 1992.

[129] H. Santoso, S. Fdida, *'Transport Layer Multicast: an enhancement for XTP Bucket Error Control'*, Proceedings High Performance Networking A. Danthine and O. Spaniol (editors), 1992.

[130] ISO/IEC, '*Common Management Information Protocol Specification*', ISO/IEC 9596-1.

[131] B. Plattner, et al., *'JVTOS: Requirements on a Joint Viewing and Tele-Operation Service'*, Deliverable RACE 2060, 1992.

[132] T. Käppner, K. Werner et al., *'The BERKOM Multimedia Teleservices Volume II Multimedia Collaboration'*, 1993.

[133] T. Crowley, P. Millazzo et al., *'MMconf: an Infrastructure for Building Shared Multimedia Applications'*, Proceedings of CSCW'90, 1990.

[134] Tina coreteam, '*Telecommunications Application Requirements'*, Document No. TB_A0.MH.002_ 1.0_93, December 1993.

[135] L. Aguilar, et.al, *'Architecture for a multimedia Teleconferencing system'*, ACM 1986.

[136] J. Rumbaugh, et al., '*Object-Oriented Modelling and Design'*, Prentice Hall, 1991.

[137] W. Veldkamp, *'EU-P208 Deliverable 2: TMN OS Platform, Analysis and definition'*, EURESCOM project P208, December 1993.

[138] A. Langsford, J.D. Moffett, *'Distributed Systems Management'*, Addison Wesley, 1992.

[139] T. Kalin, D. Barber, *'Has the OSI opportunity been fully realised?'*, Computer Networks and ISDN Systems Vol. 25, pp. 227-239, 1992.

[140] A.T. van Halteren, *'Realization of a Stream Object in ANSAware'*, Master thesis, University of Twente, Department of Computer Science (TIOS), June 1994.

[141] M. Jacobs, *'Synchronisation of audio/video in open distributed environments'*, Master thesis, University of Twente, Department of Computer Science (TIOS), 1995.

[142] Microsoft, *'Microsoft Interactive Television (MITV)'*, Architecture course, 1995.

[143] Microsoft, *'MITV system guide'*, Alpha release, June 1995.

[144] ATM monitor progress report, *'Desktop ATM where, when, why not?'*, Data communications, September 1996.

[145] Microsoft, *'Normandy'*, white paper, June 1996.

[146] R. Joseph, J. Rosengren, 'MHEG-5: an overview', www.fokus.gmd.de /ovma/mheg/rd1206.html, 1996.

[147] OMG, *'Common Object Services specifications'*, Volume 1, John Wiley & Sons, Inc, 1994.

[148] ISO/ITU, *'Interface References and Binding'*, SC21 N10747, 1997.

[149] F. Cummins, *'An Object Model for Business Applications'*, Electronic Data Systems, Troy Michigan, Presentation for OMG business object domain task force, June 1996.

[150] D. Verhoef, '*Software-ontwikkeling met componenten*', Computable, 29e jaargang, week 36, pp. 33-36, September 1996.

[151] G. Booch, I. Jacobson, J. Rumbaugh,*'The Unified Modelling Language for Object-Oriented Development'*, Version 0.8 and 0.91 addenum, Rational Software Corporation, September 1996.

[152] OMG, 'Object Analysis & Design PTF-RFP 1, Document no. 96-05-01, 1995.

[153] J.D. Williams, *'Managing Iteration on OO projects'*, Computer Innovative technology for computer professionals, IEEE Computer Society, pp. 39-43, September 1996.

[154] AFNOR comments and proposal regarding ISO/IEC JTC 1/SC 21 N9773, *'RM-ODP Enterprise Viewpont Component Standard -New Work Area Proposal'*, 1995.

[155] ITU, *'Specification and description language (SDL)'*, Z.100 Rev 1, 237 pp, June 1994.

# 9 Abbreviations

**ANSA** - Advanced Networked Systems Architecture
**ANSI** - American National Standards Institute
**API** - Application Programming Interface
**ATM Forum** - Asynchronous Transfer Mode Forum
**ATTH** - ATM To The Home
**BEO** - Basic Engineering Object
**BERKOM** - Berliner Kommunikationssystem
**B-ISDN** - Broadband Integrated Services Digital Network
**BNF** - Backus Naur Form
**CASE tools** - Computer Aided Systems Engineering tools
**CBO** - Common Business Object
**CBR** - Constant Bit Rate
**CC** - Connection Coordinator
**CD** - Committee Draft
**CMIP** - Common Management Information Protocol
**COM** - Component Object Model
**CORBA** - Common Object Request Broker Architecture
**CORDIS** - Community Research and Development Information services
**CPS** - Content Provider System
**CPU** - Central Processing Unit
**CSM** - Communication Session Manager
**DAF** - Distributed Application Framework
**DAVIC** - Digital Audiovisual Council
**DBMS** - DataBase Management System
**DCE** - Distributed Computing Environment
**DCOM** - Distributed Component Object Model
**DIS** - Draft International Standard
**DLL** - Dynamic Link Library

**DPE** - Distributed Processing Environment
**DS** - Delivery System
**DSM-CC-UN** - Digital Storage Media Command & Control User-Network
**DSM-CC-UU** - Digital Storage Media Command & Control User-User
**DSRM** - DAVIC System Reference Model
**EBU** - European Broadcasting Union
**ECMA** - European Computer Manufacturers Association
**EDI** - Electronic Data Interchange
**EIIA** - European Information Industry association
**ERO** - European Radio Communications Office
**ETIS** - European Telecommunications Informatics Service
**ETSI** - European Telecommunications Standards Institute
**EURESCOM** - European Institute for Research and Strategic Studies in telecoms
**EWOS** - European Workshop on Open Systems
**FDT** - Formal Description Techniques
**FPS** - Frames Per Second
**FTS** - Financial Transaction services
**FTTC** - Fiber To The Curb
**FTTH** - Fiber To The Home
**GUI** - Graphical User Interface
**HDTP** - Hoofd Directoraat Telecom & Post
**HDTV** - High Definition TeleVision
**HFC** - Hybrid Fiber Coax
**ICTS** - Information Communication Technologies standards
**IDL** - Interface Definition Language
**IEC** - International Electrotechnical Commission
**IEEE** - Institute of Electrical and Electronic Engineers
**IETF** - Internet Engineering Task Force
**IMA** - Interactive Multimedia Association
**IN** - Intelligent Networks
**INA** - Information Networking Architecture
**IP** - Internet Protocol

**IPC** - Inter Process Communication
**IS** - International Standard
**ISO** - International Organisation for Standardisation
**ISS** - Interactive Subscriber Services
**ISV** - Independent Software Vendor
**IT** - Information Technology
**ITU-TS** - International Telecommunication Union Telecommunication Sector
**JVTOS** - Joint Viewing and Tele-Operation Service
**KPN** - Koninklijke PTT Nederland
**LAN** - Local Area Network
**LCG** - Logical Connection Graph
**MHEG** - Multimedia and Hypermedia information object Expert Group
**MIDL** - Microsoft Interface Definition Language
**MITV** - Microsoft Interactive Television
**MMS** - Microsoft Multimedia Server
**MOD** - Movies on Demand
**MPEG** - Moving Picture Experts Group
**MPS** - Message Passing Service
**MSS** - Multimedia Systems Services
**NCG** - Nodal Connection Graph
**NMF** - Network Management Forum
**ODL** - Object Definition Language
**ODMA** - Open Distributed Management Architecture
**ODP** - Open Distributed Processing
**ODP-RM** - Open Distributed Processing - Reference Model
**OLE** - Object Linking and Embedding
**OMA** - Object Management Architecture
**OMG** - Object Management Group
**OMHEGA** - Open MHEG Applications
**OMT** - Object Modelling Technique
**OO** - Object Oriented

**OOA&D** - Object Oriented Analysis & Design
**OOSE** - Object Oriented Software Engineering
**ORB** - Object Request Broker
**OS** - Operating System
**OSCA** - Open Systems Communication Architecture
**OSF** - Open Software Foundation
**OSI-RM** - Open System Interconnection Reference Model
**PCG** - Physical Connection Graph
**PNO** - Public Network Operator
**QoS** - Quality of Service
**ReTINA** - RealTime TINA
**REX** - Remote Execution Protocol
**RFP** - Request For Proposal
**ROSA** - Race Open Services Architecture
**RP** - Recommended Practice
**RPC** - Remote Procedure Call
**SABS** - South African Bureau of Standards
**SAP** - Service Access Point
**SCS** - Service Consumer System
**SDL** - Specification and Description Language
**SNMP** - Simple Network Management Protocol
**SPIRIT** - Service Providers' Integrated Requirements for Information Technology
**SPS** - Service Provider System
**SSG** - Service Session Graph
**SSM** - Service Session Manager
**ST-2** - Stream Protocol Version 2
**TCP** - Transmission Control Protocol
**TCSM** - Terminal Communication Session Manager
**TINA-C** - Telecommunications Information Networking Architecture Consortium
**TMN** - Telecommunication Management Network
**TP4** - Transport Protocol 4

**UDP** - User Datagram Protocol
**UML** - Unified Modelling Language
**UMTS** - Universal Mobile Telecommunications System
**VBR** - Variable Bit Rate
**VCR** - Video Cassette Recorder
**W3C** - WorldWide Web Consortium
**WAN** - Wide Area Network
**WWW** - WorldWide Web
**XIWT** - Cross-Industry Working team
**XTP** - eXpress Transfer Protocol

# 10 Index

## E



## F



## G



## H



## I



## J



## K

## L



## M



## N



## O



## P



## Q

## R



## S



## T

## U



## V



## W

# 11 Excerpt

In de meeste Europese landen vindt een tweeledige verandering plaats op de telecommunicatiemarkt. Ten eerste wordt de traditionele monopolistische telecommunicatie operator die in handen is van de Staat geprivatiseerd en ten tweede wordt de markt waarop deze geprivatiseerde operator opereert opengesteld voor concurrentie van de kant van andere telecommunicatie leveranciers (waarvan sommigen zelf eveneens kort geleden geprivatiseerd zijn). Het nettoresultaat van deze veranderingen is een groeiend aantal uiteenlopende diensten en verlaging van de kosten van traditionele telecommunicatie diensten. Om effectief te kunnen concurreren moeten deze nieuwe telecommunicatie bedrijven een concurrentievoordeel vinden. Voor nieuwe diensten betekent dit dat deze van hoge kwaliteit moeten zijn, geavanceerde functionaliteiten moeten hebben en in een kort tijdsbestek geproduceerd moeten kunnen worden. Voor bestaande diensten is de voor de hand liggende keus prijsconcurrentie te voeren. De prijzen van diensten die worden aangeboden door vroegere staatsbedrijven kunnen echter aan regels gebonden worden, om het grote marktaandeel van deze bedrijven te verkleinen. In die situatie komt het accent op klantenzorg en kwaliteit te liggen. Als men dan ook nog let op de convergentie van de film- en televisiemarkt enerzijds en de videospelletjes-, computer- en telecommunicatiemarkten anderzijds, dan wordt het beeld zeer gecompliceerd.

**Probleem omschrijving**
Op deze snel veranderende markt ontstaat een probleem voor de traditionele telecommunicatie operators, omdat hun IT-infrastructuur nog niet in staat is om op doeltreffende wijze aan deze eisen te voldoen. Daar komt nog bij dat samenwerking tussen diensten die worden aangeboden door respectievelijk de computer-, telecommunicatie- en vermaaksindustrie moeilijk te realiseren is, omdat er een gebrek is aan transindustriële gestandaardiseerde interfaces. Standaardisatie-organisaties, zoals de ITU en ISO/IEC, en industriële consortia zoals de Object Management Group (OMG), de Digital Audio Visual Council (DAVIC), de Internet Engineering Task Force (IETF) en het Telecommunications Information Networking Architecture Consortium (TINA-C) zijn zich bewust van de behoefte aan standaarden om de samenwerking mogelijk te maken in een open, heterogene gedistribueerde omgeving. Er zijn aanzienlijke inspanningen verricht voor de ontwikkeling van gestandaardiseerde architecturen voor de specificatie, implementatie en inzet van gedistribueerde multimediadiensten. Door diverse oorzaken hebben deze inspanningen maar gedeeltelijk succes gehad. In de eerste plaats heeft de ontwikkeling van standaarden geen gelijke tred gehouden met de snelle veranderingen op de markt met zijn steeds toenemende concurrentie. Ten tweede zijn er concurrerende en onderling incompatibele architecturen ontwikkeld. Ten derde hebben de meeste architecturen alleen betrekking op een deel van de bestaande problemen. Er

bestaat geen allesomvattend kader waarin de verschillende standaarden geïntegreerd zouden kunnen worden.

## Doelstellingen

Op basis van de hierboven genoemde problemen wordt in deze dissertatie verslag gedaan van een onderzoek naar het gebruik van het referentiemodel voor Open Distributed Processing (ODP-RM) als allesomvattend kader voor de standaarden voor gedistribueerde multimediadiensten in open heterogene omgevingen. Dit onderzoek vereist een bestudering van de compatibiliteit van het ODP-referentiemodel met diverse standaarden. Hiertoe wordt een vergelijking gepresenteerd van de door OMG, Microsoft, Bellcore, SPIRIT, OSF-DCE, IMA, TINA-C, DAVIC en ISO/ITU-T gekozen benaderingen met betrekking tot het ODP referentie model. In deze fase van het onderzoek wordt aandacht besteed aan de vraag welke verfijningen in het ODP-referentiemodel zouden kunnen worden aangebracht om het meer geschikt te maken voor toepassing op multimediadiensten. De waarde van deze verfijningen wordt vervolgens beoordeeld aan de hand van case studies.

## Resultaten

Een overzicht van het ODP-referentiemodel wordt gegeven, met beschrijvingen van de basisbegrippen zoals object-oriëntatie, distributietransparanties en de vijf ODP-viewpoint talen. Op basis van actieve betrokkenheid bij de creatie van het ODP-referentiemodel gedurende een zestal jaren, alsmede werkzaamheden met de standaard binnen KPN Research, wordt geconcludeerd dat ODP een ideaal kader verschaft voor positionering van de verschillende aspecten van gedistribueerde systemen. ODP-RM levert ook inzicht op in gedistribueerde verwerking, waarbij duidelijk wordt dat de begrippen, regels en talen gedefinieerd in ODP-RM zeer nuttig zijn voor het modelleren en ontwerpen van gedistribueerde systemen. Maar de ODP-RM-standaard vindt desondanks geen volledige toepassing in de industrie, waar meer met de-facto standaarden wordt gewerkt (d.w.z. standaarden die buiten de officiële standaardisatie-organen om tot stand zijn gekomen) vanwege de beschikbaarheid van concrete implementaties.

Om inzicht te krijgen in de de-facto standaarden voor gedistribueerde verwerking, wordt een indruk gegeven van de betrokken consortia en van hun werkterrein en onderlinge beïnvloeding. De consortia worden met elkaar vergeleken op basis van hun benadering met betrekking tot object-oriëntatie, toegepaste scheidingsprincipes en multimediaconcepten zoals die zijn opgenomen in hun specificaties voor een Distributed Processing Environment (DPE). Dit overzicht werpt licht op de behoefte aan een algemeen kader en op de rol die het ODP-standaardisatieproces kan spelen bij het streven naar combinatie van de architecturen van de-jure standaarden (d.w.z. geproduceerd door officiële standaardisatie-organen) en de-facto standaarden. Dit soort hybridische standaardisatie zou moeten leiden tot de creatie van een langdurig in stand blijvende architectuur waarin gebruik wordt gemaakt van door de verschillende consortia geleverde componenten.

In de context van multimedia in DPE's worden de architecturen van DAVIC, MiTV, IMA en TINA besproken gezien de ondersteuning die zij bieden voor multimediadiensten. De in deze architecturen aangetroffen multimedia-concepten worden gepositioneerd vanuit de ODP information, computational en engineering viewpoints om betere vergelijking en toekomstige integratie mogelijk te maken. Op basis van dit onderzoek wordt geconcludeerd dat de verschillende consortia hun eigen (specifieke) oplossingen en concepten definiëren. Zoals te verwachten was zijn er grote onderlinge verschillen tussen deze oplossingen, vooral op het gebied van specifieke technologie. Op een hoger abstractieniveau (computational en engineering) bestaan er overeenkomsten, en alleen op dit niveau kan zinvol gebruik worden gemaakt van gemeenschappelijke terminologie. Dit betekent dat op het computational en engineering niveau multimediaconcepten gestandaardiseerd moeten worden (het leveren van interface-beschrijvingen enz.). Over de toepassing van bepaalde standaarden en specifieke technieken moet elk van de consortia dan zelf de belissing nemen.

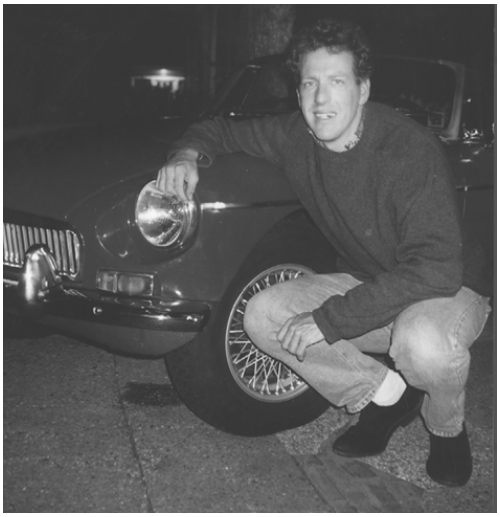
Er is bij de voorbereiding van deze dissertatie veel werk verricht voor de toepassing van de computational en engineering viewpoints op de specificatie van en ondersteuning voor gedistribueerde multimedia-diensten. In het computational viewpoint worden de ODP-begrippen 'stream interface', 'environment contract', 'binding object' en 'binding action', die ontbraken in de vroege versies van de ODP-standaard, in deze dissertatie uitgewerkt. Opneming van deze uitbreidingen in zowel de ODP-standaarden als de TINA-C documenten is voorgesteld. De raakpunten tussen deze computational concepten en engineering concepten, zoals 'stream interface reference', 'engineering channel' en 'engineering channel establishment' worden eveneens besproken. Om deze concepten te toetsen is een prototype gebouwd van een 'multiparty multimedia binding object' dat van nut is voor gedistribueerde multimediadiensten. Voor opneming van de specificaties in multimedia platforms is verdere standaardisatie nodig. Deze zal de applicatie-ontwerper in staat stellen met hetzelfde gemak complexe multimedia 'binding objects' te gebruiken als de huidige 'operational bindings'.

Naast functionele aspecten worden in deze dissertatie ook kwaliteits-aspecten (QoS) behandeld. Hiertoe wordt een algemene structuur van het begrip 'ODP environment contract' gedefinieerd met gebruikmaking van de TINA-C Object Definition Language (een uitbreiding van OMG IDL). Deze taal maakt specificatie mogelijk van de QoS-eigenschappen die specifiek zijn voor multimediatoepassingen in het computational viewpoint. Verder onderzoek is nodig voor completering van deze QoS-specificaties vanuit de vijf ODP viewpoints en voor vaststelling van de relatie tussen de ODP viewpoints. Vooral de relatie tussen de computational QoS-specificaties en de engineering specificatie (b.v. de relatie met systeemeigenschappen) vormt een uitdagend research-onderwerp.

Tot slot worden twee case studies (multimedia vergaderdienst en synchronisatiedienst) gepresenteerd ter illustratie van de in deze dissertatie voorgestelde multimedia-uitbreidingen en geïntroduceerde ontwerp methoden. Het TINA-levenscyclus model wordt als kader gebruikt voor structurering van de specificaties. Aangezien het model weinig details bevat wordt het aangevuld met de ODP viewpoint talen, de 'enterprise

template' van de ITU-T studiegroep 15 en begrippen uit de Object Modelling Technique (OMT). De case studies bevestigen de waarde van de voorgestelde uitbreidingen voor multimedia en zijn gebaseerd op onderzoek met prototypes verricht bij KPN Research.

# 12 About the author



Peter Leydekkers joined the Dutch KPN Research in 1990. He received his Masters degree in computer science from the University of Twente in 1989. His research activities include Open Distributed Processing (ODP), the TINA architecture, and the design of distributed multimedia services in an open telecommunication environments.

The author contributed to the development of the ODP reference model. He is an ITU/ISO delegate for SC21/WG7 on ODP-RM since 1991. Contributions to this standard have been reported in several articles that present computational and engineering models and provide examples on the usage of ODP-RM concepts.

During a year's involvement in the TINA Core Team in New Jersey, USA, the author contributed to the description of a general DPE that is an important part of the TINA architecture. The author's contribution include extensions to the DPE specification, in particular multimedia and telecommunication related functions. Additionally, the author contributed to TINA's Object Definition Language (TINA-ODL) and extended the language with multimedia features. These extensions have been submitted to OMG in order to extend their OMG-IDL language with multimedia specific elements.

## Publications

The information described in this thesis is partially a compilation of material earlier published in several magazines, conference proceedings, and contributions of the author to consortia and standardisation groups.

### Magazines and journals

- V. Gay, P. Leydekkers, *'ODP-RM for multimedia'*, IEEE multimedia - standards, pp. 68-73, January-March issue, 1997.
- P. Leydekkers, L.J.M. Nieuwenhuis, *'Telecommunications Information Network Architecture (TINA) Consortium'*, Telecommagazine Nr 8., October 1996.
- T. Handegard, P. Leydekkers, *'TINA Distributed Processing Environment'*, Global Communications, 1996.
- M. Jacobs, P. Leydekkers, '*Specification of Synchronisation in Multimedia Conferencing Services using the TINA Life cycle Model'*, Distributed Systems Engineering Journal 3, The British Computer Society, The Institution of Electrical Engineers and IOP Publishing Ltd, pp 185-196, November 1996.
- P. Leydekkers, L.J.M. Nieuwenhuis, *'Telecommunications Information Network Architecture (TINA) Consortium'*, Handboek Telematica, Samson Bedrijfsinformatie BV, Wolters Kluwer, 1995.
- V. Gay, P. Leydekkers, R. Huis in 't Veld, *'Specification of Multiparty Audio and Video Interaction Based on the Reference Model of Open Distributed Processing'*, Computer Networks and ISDN Systems - Special issue on ODP, Vol. 27 No. 8, pp. 1247-1262, July 1995.

### Conferences and workshops

- M. Jørgensen, P. Leydekkers, M. Mampaey, H. Yang *'Support for Distributed Multimedia Services in the TINA Architecture'*, ISADS97, Berlin, April 1997.
- M. Jørgensen, P. Leydekkers, '*Multimedia support in the TINA Architecture'*, TINA'96 conference, Heidelberg, September 1996.
- P. Leydekkers, V. Gay, *'ODP View on Quality of Service for Open Distributed Multimedia Environments'*, Proceedings of the 4th International IFIP Workshop on Quality of Service (IWQoS 96), J. de Meer and A. Vogel (Eds.), Paris, March 6-8, 1996.
- M. Jacobs, P. Leydekkers, '*Specification of Synchronization in Multimedia Conferencing Services using the TINA Life cycle Model'*, Proceedings of SDNE'95 workshop (Services in Distributed and Networked Environments), IEEE Computer Society Press, Whistler Canada, June 1995.
- P. Leydekkers, V. Gay, L.J.N. Franken, *'A Computational and Engineering View on Open Distributed Real-time MultiMedia Exchange'*, Workshop on Network and

Operating System Support for Digital Audio and Video (NOSSDAV'95), Lecture Notes in Computer Science, Springer Verlag, Hampton, NH, USA, April 1995.

- I.J.P. Kwaaitaal, P. Leydekkers, L.J. Teunissen, *'The Good, the bad and the ugly about Multimedia Conferencing Services'*, Second International Symposium on Autonomous Decentralized Systems, ISADS95, IEEE Computer Society Press, Phoenix USA, April 1995.
- A.T. van Halteren, P. Leydekkers, H.B. Korte, *'Specification and Realisation of Stream interfaces for the TINA-DPE'*, Proceedings of TINA'95 Conference, Melbourne Australia, February 1995.
- P. Leydekkers, V. Gay, *'Multimedia Conferencing Services in an Open Distributed Environment'*, In Lecture Notes in Computer Science - Number 868 - Springer Verlag - R.Steinmetz editor. Proceedings of the 2nd IEEE International Workshop on Advanced teleservices and High Speed Communication Architectures, p. 339-352, Heidelberg, September 1994.
- V. Gay, P. Leydekkers, R. Huis in 't Veld, *'Specification of Audio/Video Exchange Based on the Reference Model of ODP'*, In Broadband Islands'94 'Connecting with the End-User', North Holland-Elsevier, O.Spaniol, W.Bauerfeld and F.Williams editors. Proceedings of the International Conference on Broadband Island 94 , pp. 179-191, Hamburg, June 1994.
- P. Leydekkers, V. Gay, *'Multimedia Services in TINA-C and ODP-RM'*, IEEE Intelligent Networks '94 Workshop, Heidelberg, May 1994.
- P. Leydekkers, V. Gay , *'Open Service Architecture based on ODP : Experience in CASSIOPEIA'*, Proceedings of the Race International Conference on Intelligence in Broadband Services and Networks, Paris, November 1993.
- P. Leydekkers, L.J. Teunissen, *'Synchronisation of Multimedia Data Streams in open distributed environments'*, Lecture Notes in Computer Science, Network and Operating System Support for Digital Audio and Video, Springer Verlag Heidelberg, November 1991.

## Standards and consortia

- Contribution to the elaboration of the: Open Distributed Processing - Reference Model:

  Part 1: *'Overview'* International Standard 10746-1, ITU-T Recommendation X.901, 1996.
  Part 2: *'Foundations'*, International Standard 10746-2, ITU-T Recommendation X.902, 1995.
  Part3: *'Architecture'*, International Standard 10746-3, ITU-T Recommendation X.903, 1995.
- P. Leydekkers, N. Mercouroff, K. McKinnon, *'TINA Distributed Processing Environment (TINA-DPE)'*, Version 1.0, TR_PL.001_1.0_95, August 1995.

- B. Kitson, P. Leydekkers, N. Mercouroff, F. Ruano, *'TINA Object Definition Language (TINA-ODL) Manual'*, Version 1.3, TR_NM.002_1.3_95, June 1995.
- P. Leydekkers, M. Jørgensen, *'Stream Interfaces'*, TINA-C, contribution to OMG Telecommunications Special Interest Group (TELSIG), whitepaper, December 1995.
- ISO/IEC JTC1/SC21, *'QoS in ODP'*, Doc no. SC21 N10383, May 1996.