Telecommunications
Information
Networking
Architecture
Consortium

TINA-C Deliverable

Issue Status: Publicly Released

Overall Concepts and Principles of TINA

Version: 1.0

Date of Issue: 17th Feb. 1995

Table 1:

Page	Company	Company Contact (Address, Telephone, Fax)

Telecommunications
Information
Networking
Architecture
Consortium

Overall Concepts and Principles of TINA

Abstract: This document presents the overall concepts and principles of the TINA-C architecture. It attempts to condense into one document the key ideas of the TINA architecture to answer the question "what in essence is a TINA system". It does not give any justification, background or history for any concept or principle, as these can be found in other baseline documents. For the same reason the goals and requirements of the architecture are not given.

Keywords: Distributed Systems, Management systems, Object-orientation, Services.

Author(s):	**Martin Chapman, Stefano Montesi**
Editor:	**Martin Chapman**
Type:	**TINA Baseline**
Document Label:	**TB_MDC.018_1.0_94**
Date:	**17th Feb. 1995**

Table of Contents

List of Figures

1. Introduction

The Telecommunications Information Networking Architecture (TINA) Consortium is an international collaboration aiming at defining and validating an open architecture for telecommunications systems for the broadband, multi-media, and information era. The architecture is based on distributed computing, object orientation, and other concepts and standards from the telecommunications and computing industries.

The TINA architecture addresses the needs of traditional voice-based services, future interactive multi-media services, information services, and operations and management type services, and will provide the flexibility to operate services over a wide variety of technologies. This vision implies a software architecture that offers reusable software components, supports network-wide software interoperability, eases service construction, testing, deployment and operation, and hides from the service designer the heterogeneity of the underlying technologies and the complexity introduced by distribution.

The intention is to make use of recent advances in distributed computing (e.g., Open Distributed Processing (ODP) and Distributed Communication Environment (DCE)), and in object-oriented analysis and design, to drastically improve interoperability, re-use of software and specifications, and flexible placement of software on computing platforms/nodes. In addition, the consistent application of software principles to both services and management software is a primary goal. The TINA architecture is furthermore ensuring that a multi-supplier/provider market for telecommunications services and management systems will be possible.

1.1 Purpose

This document provides an overview of the TINA architecture as of December 1994. It specifically summarizes the work found in the other TINA-C baseline documents.

The TINA architecture is very broad in its scope and deep in its detail. It is essential to the understanding of TINA that simplifications are made, and that the main concepts and principles are extracted and presented as a whole. This document provides such an overview. The main results can be found in the TINA-C deliverable documents, of which this document is a part.

This document is not intended to provide a stand-alone tutorial on TINA. Hence the goals of the Consortium, the requirements on the TINA architecture, and other related consortium efforts are not covered. These can be found in two companion documents [1][2].

1.2 Audience

This document is intended for **managers**, new **Core Team members**, and **anyone else** who would like to understand the main characteristics of the TINA architecture.

Some background knowledge in object-orientation and distributed systems are assumed. [27] and [24] provide good introductions to these subjects.

1.3 How to read this document

Section 2 presents an overview of the organizational structure of the TINA architecture. The TINA architecture is divided into four parts: Service Architecture, Network Architecture, Management Architecture, and Computing Architecture.

Section 3 presents an overview of the general environment of a TINA system. It introduces the concept of a Distributed Processing Environment (DPE).

Section 4 presents an overview of the computing architecture of TINA. The design of systems in TINA requires the formation of four sets of specifications: enterprise, information, computational, and engineering. These are used to construct software that operates in a distributed computing environment.

Section 5 presents the main separation and layering principles that are used to define TINA systems and software.

Section 6 presents an overview of the network architecture that is assumed to be provided for TINA services. This network architecture defines generic concepts that are suitable for specialization to particular technologies, such as Asynchronous Transfer Mode (ATM).

Section 7 presents an overview of the service architecture. The service architecture defines how services should be structured, and provides models for users to access, instantiate, interact with, and terminate services.

Section 8 presents an overview of the management aspects of TINA systems. It discusses generic management principles, and provides an overview of management concerns for services, networks, and computing systems and software.

Section 9 presents an overview of the service design methodology. Use of this methodology should ensure the consistent application of the TINA architecture.

2. Organization of concepts and principles

The TINA architecture addresses a wide range of issues and provides a complex set of concepts and principles. In order to handle this complexity it is essential to partition the architecture into subsets, where each subset has a well defined scope and the relationships between subsets are well defined. This section first describes the decomposition of the TINA architecture into subsets, and then defines the relationships between them. These partitions have been derived from the main areas of concerns found in computing and telecommunications systems. The section is concluded with a brief discussion on conformance.

2.1 Decomposition of the architecture

The TINA architecture is decomposed into four main subsets:

- The **service architecture** defines a set of concepts and principles for the design, specification, implementation, and management of telecommunication services.
- The **network architecture** defines a set of concepts and principles for the design, specification, implementation, and management of transport networks[1].
- The **management architecture** defines a set of concepts and principles for the design, specification, and implementation of software systems that are used to manage services, resources, software, and underlying technology.
- The **computing architecture** defines a set of concepts and principles for designing and building distributed software and the software support environment.

In addition to these subsets, the **overall architecture** contains the generic concepts and principles that should be applied to the design, specification and implementation of any type of software system in a TINA consistent way. In particular, the overall architecture contains separation principles that should be observed. The overall architecture is derived by extracting, and generalizing where appropriate, the common principles found in the four sets. Note that this document defines the overall architecture.

Each of the four architecture sets are further decomposed into more detailed subsets of concepts and principles. This decomposition structure of the TINA architecture is depicted in Figure 2-1. In this figure, the further decomposition of the above four sub-sets is shown. These are examples and do not form an exhaustive list.

2.2 Relationships between the architectures

The overall architecture defines the common concepts and principles to be applied. The service, network, and management architectures must be consistent with the overall architecture.

1. The term transport network encompasses both transmission and switching technology.

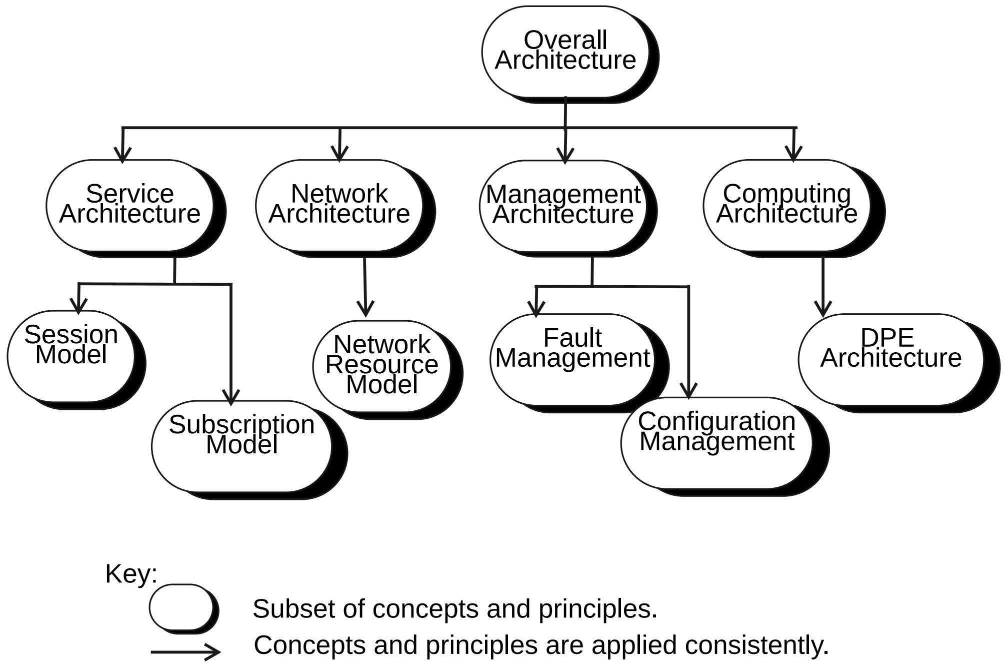


Figure 2-1. Decomposition of the TINA architecture

The computing architecture must be consistently used within the service, network, and management architectures. This will ensure that software is constructed following the same basic principles, and will result in interoperable and portable software. The management architecture is specialized within the computing architecture for the purpose of managing the computing systems and software.

The service architecture requires service oriented abstractions of network resources. The network architecture provides these abstractions, providing services oriented views of connection establishment, modification, and release. The management architecture is specialised in the service architecture for the purpose of service management.

The network architecture provides service oriented abstraction of network resources that the service architecture relies upon. The management architecture is specialised in the network architecture for the purpose of network management.

The management architecture defines the generic concepts and principles suitable for the management of services, networks, and computing infrastructures. The management architecture must therefore be consistently applied, and specialised, within the service, network, and computing architectures. Management software, that has users interacting with it to perform management tasks, should be offered as services, and hence the concepts and principles of the service architecture can be applied.

Figure 2-2 depicts the relationships between the architectures. Note that when one architecture relies on another, requirements are being made. These general requirement relationships are necessary to ensure that each architecture provides a suitable set of concepts and principles.

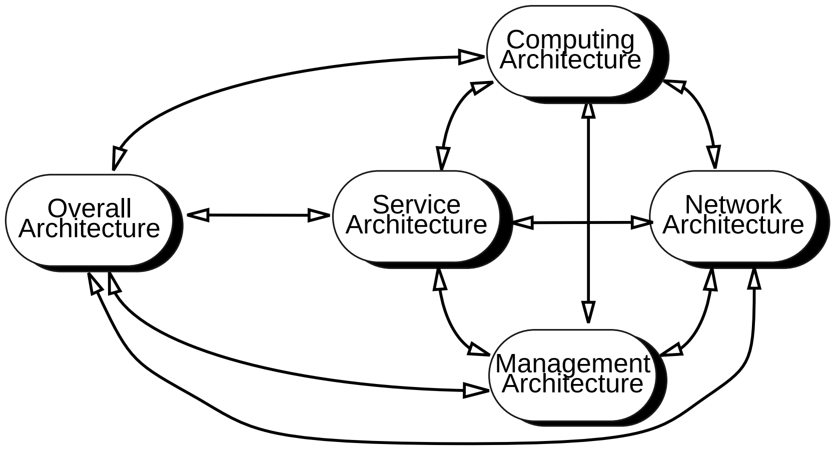


Figure 2-2. Relationships between architecture subsets

2.3 Conformance

The TINA architecture is expressed as a set of concepts and principles, and defines the style and nature of software in a TINA system. These concepts and principles are embodied, at a more precise level of detail, as a set of object and interface specifications. The object and interface specifications are subject to principles such as constraints that govern which objects should use other objects, or procedures that should be followed during the design, deployment, and operation of TINA conformant software. Some TINA specifications will be submitted to standards bodies. Others will be examples of how the concepts can be applied, but will not be required for TINA conformance.

Considering the above, the concepts and principle of TINA are classified as being of one of two types: architectural and internal. Architectural concepts and principles should be applied to the design, specification, and implementation of any TINA system by anybody. Architectural concepts may be further classified as being external. External concepts and principles define those aspects that must be adhered to in order to claim a product is TINA conformant. Internal concepts and principles on the other hand are used and applied by the TINA-C Core Team and prescription of their use outside this team is entirely optional. An example of an internal principle is the use of a form of GDMO as one of the specification languages; the TINA architecture is independent from any language and notation, however, one has to choose one in a team in order to get precise specifications. Internal concepts and principles will be clearly marked, all others will be architectural. Currently no external concepts and principles (conformance/reference points) have been defined in TINA.

3. Overall framework of TINA systems

This section describes the basic features of the environment that the TINA architecture is expected to affect. The purpose of this section is to provide the readers with an understanding of the scope of the architecture in terms of telecommunications products and life-cycle stages. In order to describe properly the scope of the architecture, some concepts of layering are introduced here.

3.1 Basic characteristics of TINA systems

The TINA architecture provides a set of concepts and principles to be applied in the specification, design, implementation, deployment, execution, and operation of **software** for telecommunication systems. A telecommunication system includes a collection of hardware and software resources that are able to provide services to different stakeholders[1], either directly or indirectly through other systems. Telecommunication software is used here in its broadest sense and encompasses all software for operating, maintaining, and utilizing telecommunication services and transport networks. A transport network includes switching and transmission equipment.

The range of services provided by a TINA system to stakeholders is potentially wide, and includes voice-based services, interactive multi-media services, information services, and management services. Services types are intentionally left open so as not to restrict the application of the architecture to the provision of certain classes of services. Similarly, the types of network technologies that can be used are not restricted by the architecture.

A telecommunications system, from a TINA perspective, does not necessarily coincide with administrative boundaries. In general, a system will be constructed by the joining, or federation, of the hardware and software resources of different administrative domains (e.g. several network operators, service providers, and customers).

The notion of telecommunication system defined here includes Customer Premise Equipment. Unlike other architectures and services, the TINA architecture does not hide CPEs behind a traditional User Network Interface (UNI). Rather it is accepted that software running on CPEs may be part of an overall system or service design. TINA provides more broader concepts and principles for CPE and non-CPE interaction, which may include - but is not restricted to - traditional UNIs.

The final characteristic worth noting is that telecommunication systems conforming to TINA will not exist in isolation. There are many other systems in existence, and these will not disappear overnight. Thus TINA systems may have the ability to interwork with non-TINA systems, at either (or both) the network and service levels. Currently no architectural aspects for interworking have been defined.

1. Stakeholders involved are **customers**, **end users**, **service providers**, **network providers**. See [6]

Figure 3-1. summarizes the characteristics of TINA systems. It shows that a telecommunications system consists of services and networks provided to different stakeholders, and that it can be divided into several administrative domains. TINA systems can interact with networks and services provided by other (non-TINA) systems.

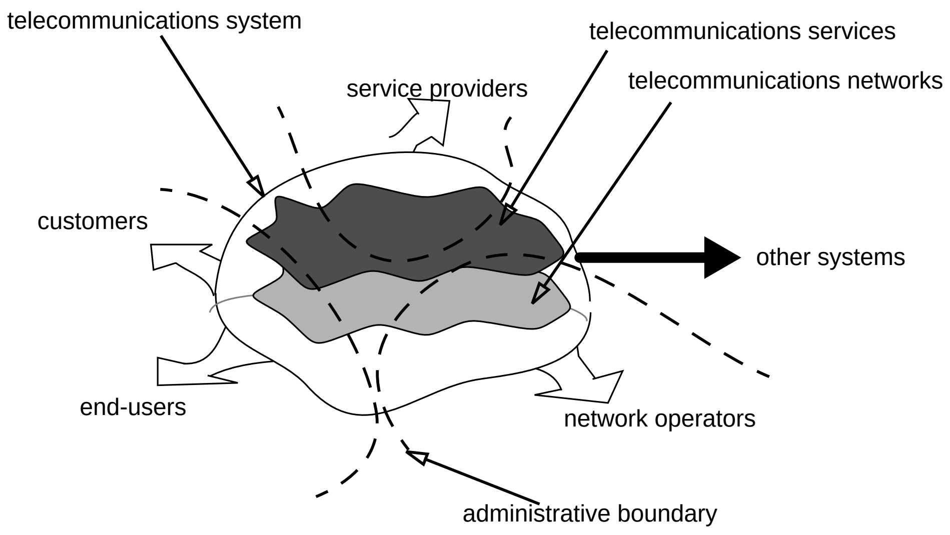


Figure 3-1. Characteristics of a telecommunications system

3.2 Software in TINA systems

TINA defines a software architecture for telecommunication systems. A software architecture defines concepts and principles of the structuring of software, and for the constraints that should be applied during the specification, implementation, execution, and operation of software. TINA defines the basic structuring and operation environment for software.

There are two basic principles that are observed:

- Telecommunications software is considered as a, potentially large, distributed software system, to which distributed computing techniques can be applied
- Object-oriented software techniques can be applied

Considering these two principles a rough structure of telecommunications software emerges. This structure, shown in Figure 3-3, features the separation between **telecommunications applications**, that is the software actually implementing the capabilities provided by the system, and a **Distributed Processing Environment** (DPE), that is software supporting the distributed execution of telecommunications applications. The telecommunications applications are designed and implemented as sets of interacting objects. Similarly a DPE is designed in an object-oriented way. Although both applications and a DPE are designed using object-orientation, the architecture does not require that they are implemented using object-oriented programming languages, although doing so may lead to a more consistent system.

In order to define more precisely the computing structure, Figure 3-3 depicts a layered architecture. At the bottom are the hardware resources, such as processors, memory, and communication devices. Above this is a software layer that contains the operating systems, communications, and other support software found in computing systems. This layer is called the **Native Computing and Communications Environment** (NCCE). Above this is the DPE layer, followed by the telecommunications applications layer.

Figure 3-3 also shows that a TINA system may interwork, at different levels, with non-TINA-systems. TINA applications can interwork with other (non-TINA) telecommunication applications, TINA DPEs can interwork with non-TINA DPEs, and NCCEs in TINA systems can link with NCCEs in other systems. It is implied, but not shown in the figure, that non-TINA software coexists alongside the DPE and applications layer.

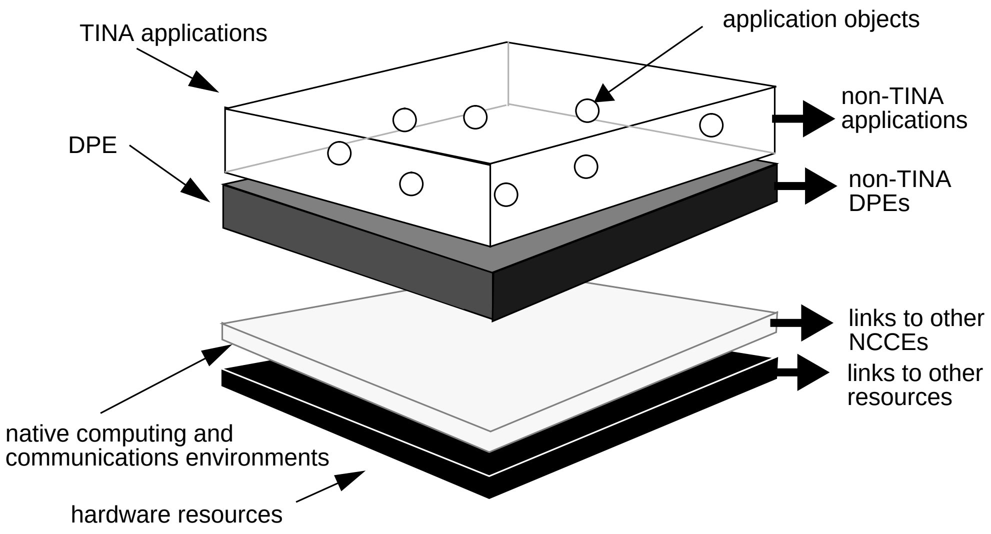


Figure 3-2. Basic structure of telecommunications software in a TINA environment

For simplicity, Figure 3-3, shows the NCCE as a homogeneous layer. In reality it is made up of a set of interconnected computing nodes, where each node may support different hardware and software technology. An example is a node that supports the UNIX[2] operating system and provides an Open Systems Interconnection (OSI) protocol stack. The DPE provides a technology independent view of computing resources, allowing technology dependent aspects in applications software to be minimized. This promotes easier design, software re-use, and portability.

2. UNIX™ is a trademark of AT&T.

The DPE has another important function. It shields from applications the distributed nature of the system. Applications are written as a set of interacting objects. The objects of an application may, for operational reasons, be located on different nodes from each other. The DPE provides support for object location and remote interaction, and allows applications to be designed without knowledge of the eventual locations of its parts. This promotes easier designs and software reuse.

The design and implementation of the NCCEs is dependent on the specific network node and is outside the scope of the architecture. TINA only concerns itself with the DPE and the applications layer. Saying this, it is worth noting three points on the boundary between the DPE and the NCCE:

1. Non-TINA software may coexist, on a node, with the DPE and TINA application software. TINA does not concern itself with this software.
2. The DPE requires the use of protocol stacks, and has requirements on their capabilities. Protocol stacks provided by an NCCE may meet these requirements. However if appropriate stacks are not provided by the NCCE, they must be provided in the DPE layer.
3. Some nodes may come equipped with special hardware resources, such as video processing boards, switch fabrics, and speech recording/playback devices. The software for these devices are not considered part of the NCCE for the simple reason that they do not perform general purpose computing or communications functions. The software associated with these devices may be considered part of the TINA application layer, if they are made available to other TINA applications.

3.3 Distributed environment

In order to provide a more comprehensive description of the above described environment it is worth elaborating to give some insight to the distributed appearance of the telecommunications system.

Figure 3-3 shows different nodes in a system, each equipped with its NCCE of varying complexity: some nodes may have a minimal environment, some others may feature a sophisticated environment. The figure also shows that different implementations of the DPE may exist on different nodes, where diversity is due both for technical reasons (coping with different technologies) and for market reasons (different vendors). All these implementations, in principle, shall present to the telecommunications applications the same capabilities, as shown by the “DPE surface”. The nodes that comprise the system may be owned by different administrations, including customers, service providers and network operators. A multi-supplier DPE environment, in which the DPEs on each node may come from different vendors, requires an agreed inter-DPE interface to allow the DPEs to work as a whole (provide the “surface”).

Nodes of a system are divided into two categories: **DPE nodes** and **non-DPE nodes**. Quite simply, nodes in the first category host DPE software, while the other nodes do not. In general it is assumed that telecommunications applications run on top of the DPE. However, it is possible to place TINA software on non-DPE nodes provided it is possible for application software on DPE nodes to interact with it *as if it were* on a DPE node i.e. the non-DPE node

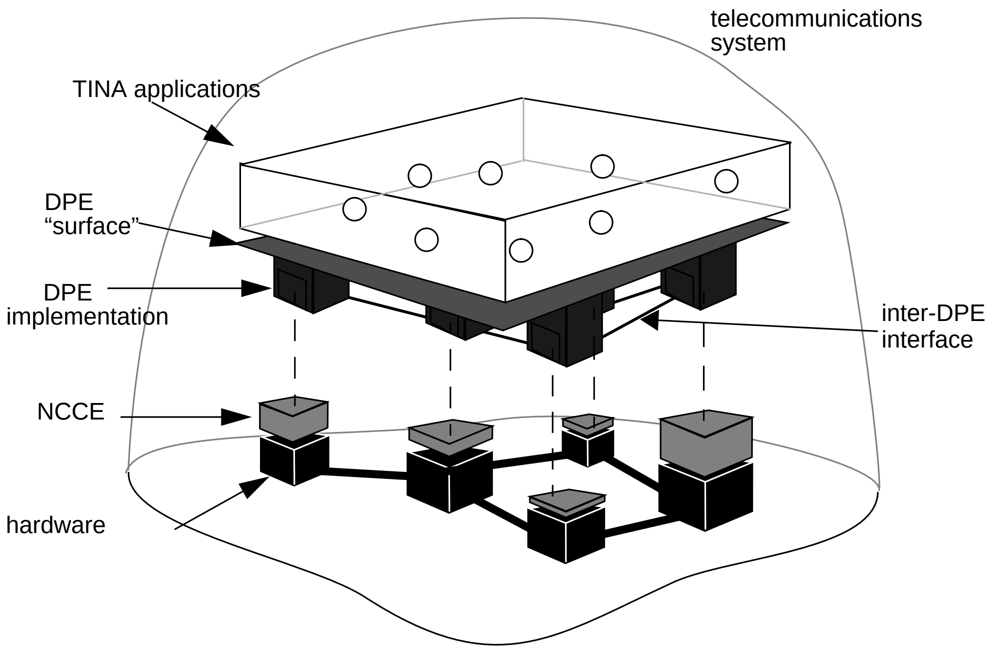


Figure 3-3. Underlying nodes of TINA systems

should support, either directly or through a gateway, an inter-DPE interface. Software on non-DPE nodes will, however, not benefit from the technology independence and portability support provide by the DPE.

Non-DPE nodes include nodes without processing capabilities, such as a telephone handset, and nodes provided with general purpose processing capabilities, where DPE software is not deployed because of network design decisions, or because of existing software (legacy systems). Examples of possible DPE nodes are service nodes, switching nodes, administration nodes, and workstation-based CPEs. Although these examples are provided, the TINA architecture does not prescribe what application software goes into a node.

Completing the story, Figure 3-3 shows a structure in which the DPE and application software is divided up and allocated to nodes. This does not contradict the former depictions as the DPE software allocated to different nodes will work together to give the impression of one homogenous DPE.

3.4 Telecommunications services and networks

The previous discussion focussed on a description of software and an environment for its execution. The layering architecture presented is a very generic (distributed) computing architecture, and is indeed not specific to telecommunications. Telecommunication specific concerns are introduced by considering the types of application software. In telecommunications, software concerns itself with the provision and operation of services and the operation of transport networks.

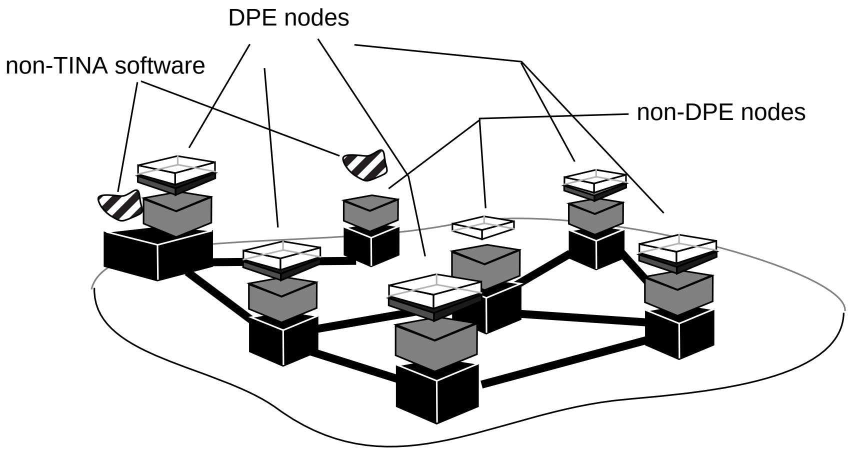


Figure 3-4. Full distributed view of TINA systems

Transport networks provide a set of switching and transmission resources which may be used and managed by software in the application layer. Figure 3-3 shows the addition of the transport network to the basic environment. Also shown is the software that controls and manages the transport network.

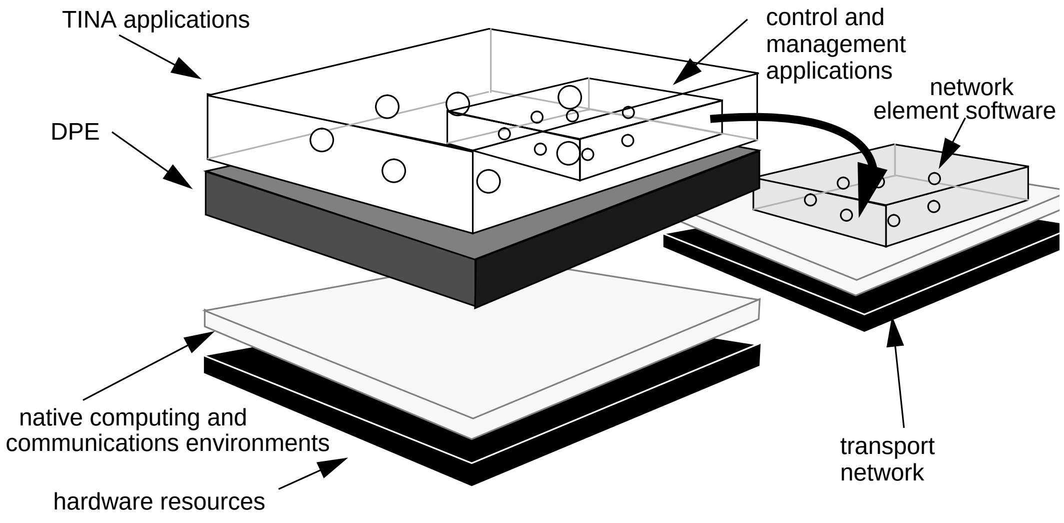

TINA applications
control and management applications
network element software
DPE
native computing and communications environments
hardware resources
transport network

Figure 3-5. Transport network in the environment

The transport network will comprise network elements that may contain computing engines. TINA makes no assumptions as to whether there will be software on the network elements that make up the transport network. TINA also does not require the presence of a DPE on these elements. Since a DPE may or may not be present, TINA does not insist that software on these network elements is structured according to the concepts and principles of the TINA architecture. Consequently, communication between application software, such a management software, and non-TINA software on network elements is not covered by the TINA architecture. It is a proprietary matter how TINA-software communicates with non-TINA software residing on network elements. It is worth stressing that although there is no requirement for a network element to have a DPE, it can have one if operational needs dictate, and thus could be capable of hosting TINA applications. In these cases however, it is still out of the scope of TINA to consider how software can manipulate the underlying hardware.

3.5 Software life-cycle

Prior to the more detailed discussion of following sections it should be observed that the figures above do not fully describe the scope of the architecture since they account for only some stakeholders playing certain roles i.e. end-users, customers, network operators, service providers. They do not account for other stakeholders that are involved in TINA environment, such as Service or Network Designers and Developers. These stakeholders are not shown because those figures implicitly looks at the run-time view of the system. To provide a complete architecture for telecommunications, it is necessary to consider the complete life-cycle of software.

The life-cycle model identifies five major phases for software in a telecommunications environment (Figure 3-6). The Need phase covers the identification and analysis of the reasons to introduce new software. The construction phase covers the design, implementation, and testing of software. Deployment covers the introduction of software into the system. Operation covers the execution, utilization, and run-time management of the software. Withdrawal covers the extraction of the software from the system. The life-cycle model does not imply a sequential ordering of Phases. There may be many activities associated with a phase, and pure sequencing may not be appropriate. The life-cycle model is described in [11].

This model is generic enough to accommodate differences in the life-cycles of different types of software, such as DPE and application software. It can be specialized by decomposing each phase into sub-phases. An example specialization is the Service Life-cycle model.

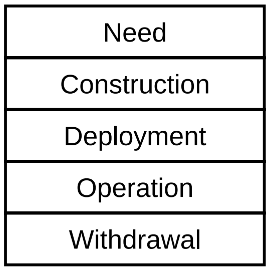


Figure 3-6. Phases in the software life-cycle model

4. Computing architecture

The computing architecture define the modelling concepts that should be used to specify object-oriented software in TINA systems. It also defines the distributed processing environment (DPE) that provides the support system allowing objects to locate and interact with each other. These concepts are based on the Reference Model for Open Distributed Processing (RM-ODP) [24][25][26]. RM-ODP is a standard for the definition of generic, non-telecommunication specific, distributed systems. The computing architecture refines, and adapts the RM-ODP standard, so that it is suitable for the design of telecommunication systems.

A telecommunications system is very complex. In order to understand this complexity the system can be specified from five viewpoints. Each viewpoint focuses on a sub-set of the characteristics of a system. The complete description of the system can be obtained by examining the specifications of all the viewpoints.

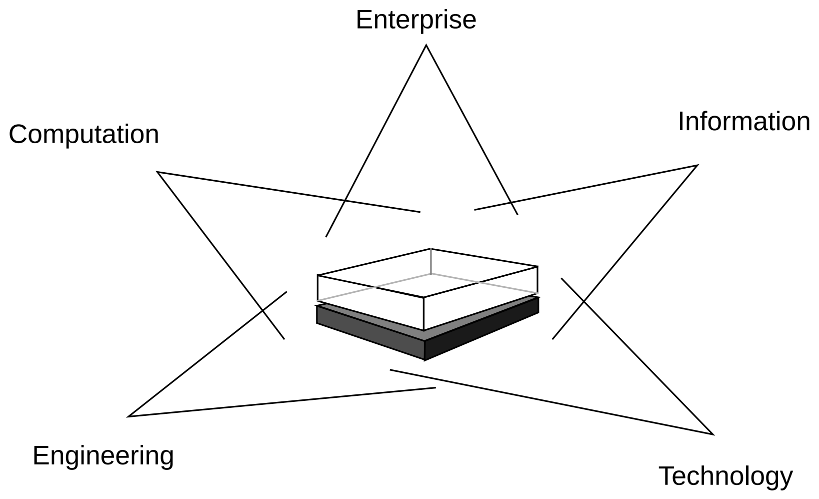


Figure 4-1. Viewpoint separation

The five viewpoints are as follows (Figure 4-1):

- **Enterprise** viewpoint focuses on the purpose, scope and policies for the system.
- **Information** viewpoint focuses on the semantics of information and information processing activities in the system.
- **Computational** viewpoint focuses on the decomposition of the system into a set of interacting objects which are candidates for distribution.
- **Engineering** viewpoint focuses on the infrastructure required to support distribution.
- **Technology** viewpoint focuses on the choice of technology to support the system.

The TINA architecture defines modelling concepts for the first four viewpoints. Modelling concepts define abstract language constructs[1] that a can be used to express the design of a system.

4.1 Enterprise modelling concepts

Enterprise modelling concepts provide a framework for building enterprise specifications or models. An enterprise model describes a system from the perspective of the organizations and people that will use or operate the system.

The enterprise modelling concepts include:

- Stakeholders, which are used to define the actors, or agents, involved in a system
- Roles, which are used to describe the types of activities a stakeholder can partake in
- Requirements, which are used to express desired features and capabilities of a system
- Obligations, which are used to describe the responsibilities of the stakeholders involved
- Policies, which sets constraints on the use and operation of the system.

Stakeholders and roles are described in [11]. Generic requirements on TINA systems can be found in [2]. Obligations and policies have not yet been elaborated in TINA.

4.2 Information modelling concepts

The information modelling concepts provide the framework for information specifications. An information specification describes the problem domain that the system will be covering [3].

The information modelling concepts are comprised of:

- Information bearing entities (**information objects**)
- Classification of information objects into **object types**
- **Relationships** between the entities, and
- Constraints and rules that govern their behaviour, including the rules for their creation and deletion.

When building an information specification, the designer should focus on **what** the system is doing, without regard to the how the system is provided. The designer of an information specification should not define the pieces of software that are required, nor on what nodes software should reside.

1. An abstract language construct is independent from any notation (syntax) used to represent it.

A notation chosen for information specifications is a form of **GDMO** (Guidelines for the Definition of Managed Objects) [22] with **GRM** (General Relationship Model) [23]. GDMO and GRM was chosen because it is widely used in the telecommunications management community and there is a large body of GDMO specifications available for reuse. Standard GDMO-GRM allows for many concerns to be expressed in a specification, including information and computational issues. TINA uses only the aspects of GDMO and GRM that are suitable for (TINA) information modelling. This notation is called quasi GDMO-GRM (or q-GDMO/GRM for short). It is worth noting the existing GDMO-GRM specifications have to be reworked to comply with the TINA information modelling concepts. The OMT (Object Modelling Technique) graphical notation [27] has been adopted for the diagrammatic representation of information specification. Figure 4-2, highlights the main constructs of this notation. A tool has been constructed that translates OMT diagrams into quasi GDMO-GRM [10].

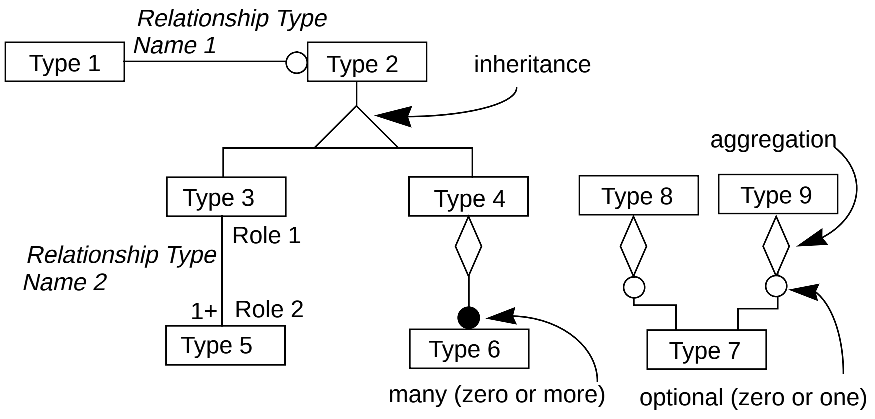


Figure 4-2. Extracts of the OMT graphical notation

The use of quasi GDMO-GRM and OMT are internal (to TINA-C) constraints. Any notation that can express the information modelling concepts can be used.

4.3 Computational modelling concepts

The computational modelling concepts provide the framework for computational specifications. A computational specification describes distributed telecommunications applications in terms of software entities called **computational objects** [4]. Computational objects interact with each other to provide the application.

The TINA computational modelling concepts defines the rules of how **computational objects** interact with one another. Computational objects are the units of programming and encapsulation. Objects interact with each other by sending and receiving information to and from **interfaces**. An object may provide many interfaces, which may be of different types.

There are two forms of interface that an object may offer or use: **operational interface** and **stream interface**. An operational interface is one that has defined operations, that allow for functions of the offering (server) object to be invoked by other (client) objects. An operation may have arguments and may return results. A stream interface is one without operations (i.e., there is no notion of input/output parameters, requests, results, or notifications). The establishment of a **stream** between stream interfaces allows for the passing of other structured information, such as video or voice bit streams. Streams are established by interacting with service and network components, defined in the network and service architectures (Section 6 and Section 7). Figure 4-3 depicts these concepts.

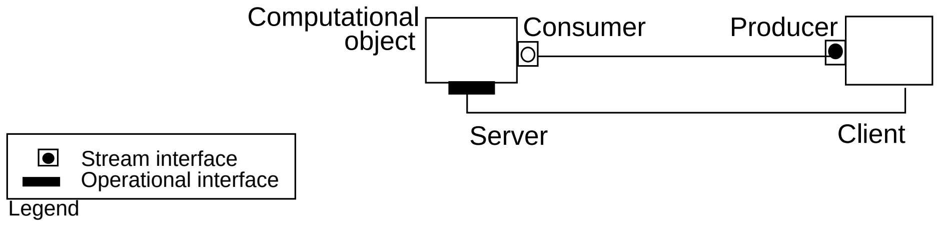


Figure 4-3. Computational modelling concepts

When building a computational specification of a system, the designer should focus on the programming entities (objects) required, what interfaces they offer, and what interfaces of other objects are required. The computational specification of a system should be compatible with the semantics of the information specification of the same system. Computational designers should not concern themselves with the distribution aspects of the software i.e. where the objects will be located.

The notation chosen for computational specifications is called TINA ODL (Object Definition Language). ODL enhances OMG IDL (Object Management Group Interface Definition Language) [28]. An example enhancement is the specification of objects that are made up of one or more interfaces; OMG IDL does not have the notion of objects with multiple interfaces.

Like quasi GDMO-GRM, use of the ODL notation is an internal (to TINA-C) constraint. Any notation that can express the computational modelling concepts can be used.

4.4 Engineering modelling concepts

The engineering modelling concepts provides the framework for describing the deployment view of an application and on the organization of an abstract infrastructure, called the *Distributed Processing Environment* (DPE). These concepts provide a means to specify the structure of a distributed TINA application in terms of the components that are actually distributed on a set of computing nodes, and the models and mechanisms to support their execution and interaction [5].

The engineering modelling concepts describe how to deploy computational objects for execution on the infrastructure. The concepts for deployment include three engineering units:

- **DPE Node**, which is the engineering abstraction of a unit of resource administration providing support to a DPE.
- **Capsule**, which is a subset of a node and models a unit of resource allocation and encapsulation.
- **Cluster**, which defines a group of co-located objects. The objects in a cluster are required to migrate and be activated together.

The engineering modelling concepts that define the DPE architecture include the concepts of DPE kernel, kernel transport network, and DPE services.

The **DPE kernel** provides support to object-life-cycle control and inter-object communication. Object life-cycle control includes capabilities to create (instantiate) and delete objects at run-time. Inter-object communication provides the mechanisms to support the invocation of operations provided by operational interfaces of remote objects[2]. The DPE kernel provides basic, and technology independent, functions that represent the capabilities of most computing systems (i.e. the ability to run programs and the ability for programs to communicate with each other). A DPE kernel is assumed to be present on all nodes that contain a DPE.

To facilitate communication between remote objects, the DPE kernels on different nodes communicate with each other. This communication is achieved through the **kernel transport network** (KTN). The kernel transport network provides a technology independent view of the communication facilities provided by the NCCEs of the DPE nodes. The KTN is a virtual network that is logically different from the transport network.

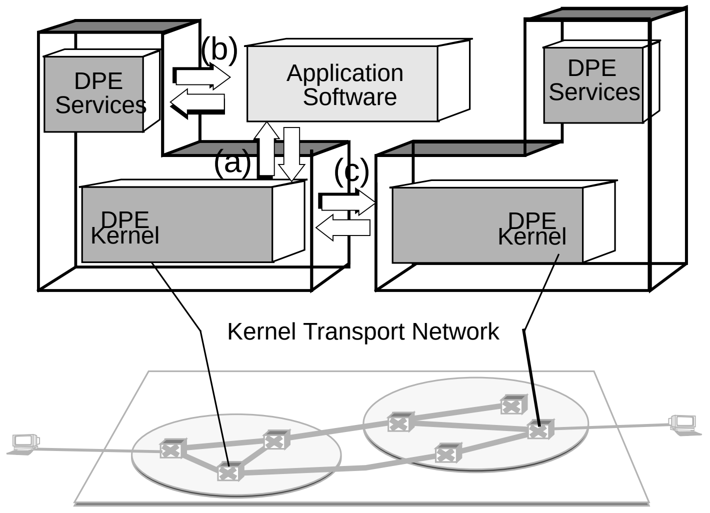
DPE services provide operational interfaces to support for the run-time execution and communication of objects. Included are trading services, that provide for the run-time location of remote object interfaces, and notification services, that provide for the emission of typed notification messages to interested objects.

A distinction is made between DPE kernel and DPE services because it is necessary to define basic capabilities that should be provided on all DPE nodes, from advanced capabilities that may be present on fewer nodes.

Figure 4-4 depicts the DPE architecture. Interface (a) depicts the interface to the DPE kernel. Interfaces (b) depicts the interfaces to DPE services. Interface (c) depicts an inter-DPE interface to facilitate communication between objects on different nodes.

At present, no notation is used in TINA for engineering specifications.

2. A remote object is one that is in a different cluster. Even though clusters may be on the same node, the possibility exists for a clusters to migrate to different nodes during run-time. Thus assumptions on the locations of clusters should not be made, and inter-cluster communication should use remote invocation mechanisms.



Key:

(a) Basic DPE interface
(b) DPE services interface
(c) Inter-DPE interface

Figure 4-4. Basic DPE architecture

5. Architectural layers and separations

This section outlines some layering and separation principles found in the TINA architecture.

The purpose of layering and separation principles is to be able to partition the problem space into different areas of concern. The effect of separation principles is to allocate objects in a design to particular areas. Entities are allocated to an area based on the function and/or role they perform, and on their requirements to interact with other objects.

In general, separation principles are orthogonal to the viewpoints defined in the computing architecture.

5.1 Main separations

The separations identified in TINA are based on two layering principles: computing and management layering.

Computing layering consists of hardware, operating system, DPE, and application layers, as described in Section 3.

Management layering, as defined by TMN (Telecommunications Management Network), consists of network elements (NE), element management layer (EML), network management layer (NML), service management layer (SML), and business management layer (BML) [21].

The TMN layering has been generalized in TINA. The TINA architecture defines three layers viz service, resources, and element (explained below). The business management layer has not been addressed by TINA. The TMN layering has been adapted for two reasons. Firstly, the layering has been generalized so as to allow for the management of many different types of elements. TMN intentionally covers only the management of transport networks. In TINA systems, other types of elements need to be managed. Examples are, software, computing nodes, and the DPE. Secondly, the TMN layering, when slightly modified, is also suitable for the partitioning of non-management software. For example, a service logic that manipulates resources to perform its function fits naturally with the service and network (resource) split.

The management layering principles are used to partition the software in the application layer. Figure 5-1 depicts a refined version of Figure 3-3, showing the separations in the applications layer.

5.1.1 Element layer

The element layer is populated by objects that represent atomic units of physical or logical resources, defined for allocation control, usage and management purposes. Objects in the element layer are called elements. From a TMN perspective, elements are proxy representations of physical equipment found in transport networks, and the element layer provides NEL functions. Examples include switching fabrics and transmission equipment. An ele-

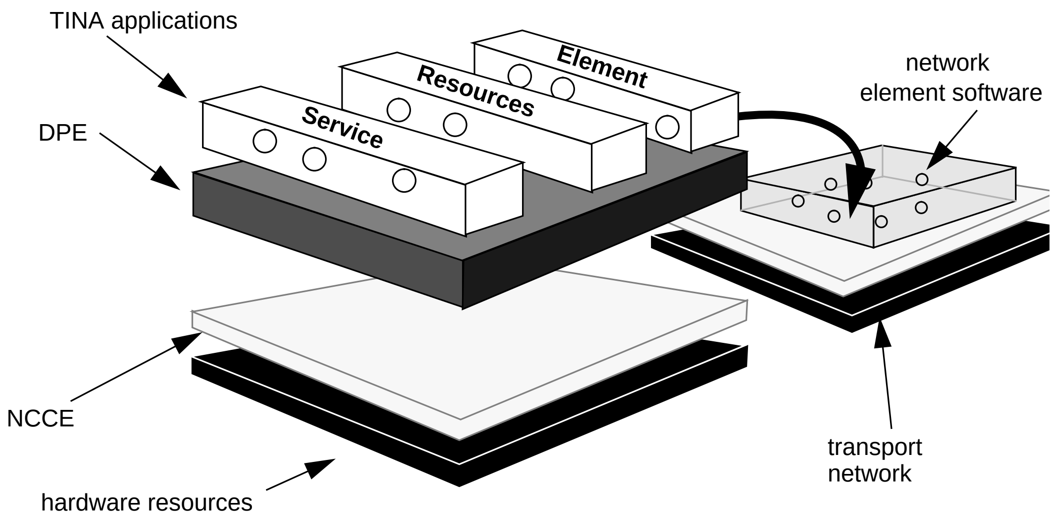


Figure 5-1. Layers and separations in TINA

ment that is a proxy for physical equipment, is a unit of software that complies with the computing architecture e.g. is realized by a computational object. Other TINA software may interact with the element in a TINA consistent way. The element is responsible for communication with the actual device (network element software), which may follow proprietary or standards based protocols and interfaces. As mentioned in Section 3-3, the way an element interacts with the actual equipment is outside the scope of TINA.

Elements that represent physical devices are necessarily technology dependent. However, the use of 'standard' element representations could provide vendor independence. For example, a Synchronous Digital Hierarchy (SDH) switch is very different from an Asynchronous Transfer Mode (ATM) switch. One cannot not substitute an SDH switch with an ATM switch, even if abstractly they are both switches with similar characteristics. One can, however, substitute an ATM switch from one vendor with an ATM switch from another if the same element definition is used by both vendors.

Elements are not restricted, as in TMN, to representing only physical entities. Logical resources can also be elements. An example is an element that represents a unit of software. The element in this case will provide interfaces to allow management software to manipulate the status of the software, such as to control its availability.

The element layer provides a view of individual resources. In TMN parlance, elements are similar to managed object definitions. From a management perspective, here is no managing functionality in the element layer, only representations of the things to be managed. The relationships between elements, such as a transmission line attaches to a switch fabric, is not represented in the element layer. In the element layer, elements do not directly interact with each other.

5.1.2 Resources layer

The resources layer contains objects that maintain views and manipulate collections of elements and their relationships. It also provides the service layer with abstract representations of elements.

From a TMN perspective, the resources layer corresponds to the Element Management Layer and Network Management Layer. The resources layer will contain objects that contain the managing functions to be applied to elements, either individually (element layer equivalence) or as a group (network layer equivalence). The main reason why the element and management layers in TMN have been combined in the resources layer is that the management functionality in EML and NML are very similar, only the scope, individual element or collection of elements, is different. Because of this, common management and support software can be defined. An example is the management of faults. The handling and processing of faults from an individual element or from a collection of related elements is very similar, and a common fault manager could be defined in the resources layer.

The managing of elements can be dependent on technology. Management software would have to be tailored for the management of ATM switches or SDH switches, for example. This lowest sub-layer of management software is therefore technology dependent. The resources layer is responsible for providing technology independent views of elements, so as to be suitable for services. In fact, it is common that services do not see individual elements, rather services are provided with technology independent abstractions of collections of elements. An example is an abstract notion of ‘connection’ that can be translated into ATM technology.

It is worth stressing that the resources layer is not restricted to the control and management of elements in the transport network. Anything that is represented in the element layer should be handled by the resources layer.

5.1.3 Service layer

Service layer is populated by objects involved in the provision of services to stakeholders. Objects in this layer are either specific to a given service or are service-independent. The first embody logic, data, and management capabilities specific to a service. The second provide generic service access, control and management capabilities.

5.1.4 Interaction constraints

The layering principles defined above should not be used to strictly classify objects. This means that an object may be in more than one layer provided the purposes are different. For example, an object that provides a service logic in the service layer, could also be considered an element for the purposes of managing the object as a piece of software, such as migrating it from a node that cannot meet processing demands.The layering principles should therefore be applied relative to a particular problem.

From a particular problem to be solved (offering a service, managing software, managing networks, for example), the separation of objects in the three layers should be applied. In general, objects are expected to access only objects in their same layer or in a contiguous layer (i.e., service objects would request operations only to objects in the resources layer). In specific cases, which should be carefully identified and specified, this constraint might be relaxed.

5.1.5 Multi-suppliers and operators

In general, a TINA system will consist of software and hardware from multiple suppliers, and the provision of services and networks will be through a cooperation of service and network providers. Objects owned by one party may interact with objects owned by another party. This may occur within and between layers. An example is a third party service provider who does not own a network. The third party service provider provides service logic that will manipulate the networking resources via the resources layer owed by another party.

5.2 Separations from a service perspective

A service, once deployed into a network, may have different people performing different roles with respect to the service. The end-users, in the customer environment, will interact with a service in order to obtain the effects of the service (i.e. what that service has been defined to provide). Service managers will interact with the service in order to provide subscriptions, control service usage, and to provide charges and bills. Network managers will be involved in order to provision the resources and ensure resource availability required for a service. People performing service and network management functions may belong to customer environments as well as Service and Network Provider ones. These roles are not exhaustive.

A service must be designed to suit the needs of all the different roles that people might play with respect to it, such as, end-user, service manager and network manager. The roles that need to be considered should be captured as part of the requirements on the service.

When a service is placed into a network a number of considerations must be made with respect to functionality and to what data/information is required. It is important that all aspects of a service are incorporated into a design and deployed at the same time. Four areas should be covered: access to the service, the service core (service logic), management of the service, and the resources (and their management) required to run the service.

Figure 5-2 shows a generic service model, as defined by ROSA [16]. It incorporates the separation of a service into the different areas of concern (access, core, management and resources) and how these relate to different roles that may be played with respect to a service. The areas of concern in this figure define separation principles for service components. A service in a TINA system should incorporate aspects of access, core, management and resources for all the roles required to be supported.

Figure 5-3 shows a generic access model for services (also from ROSA), where additional objects may be required to support the interaction with multiple services at the same time.

These separation principles are used by the service architecture. They are not just relevant to the services layer, since some of the separations can be related to objects in the resources or element layer.

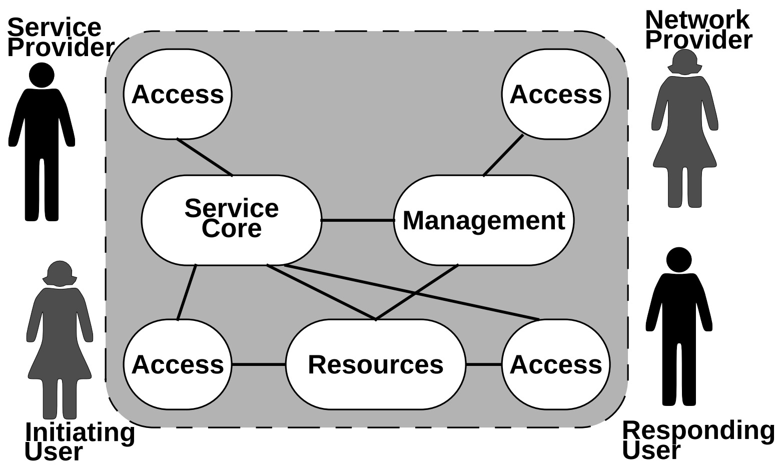


Figure 5-2. Generic Service Model

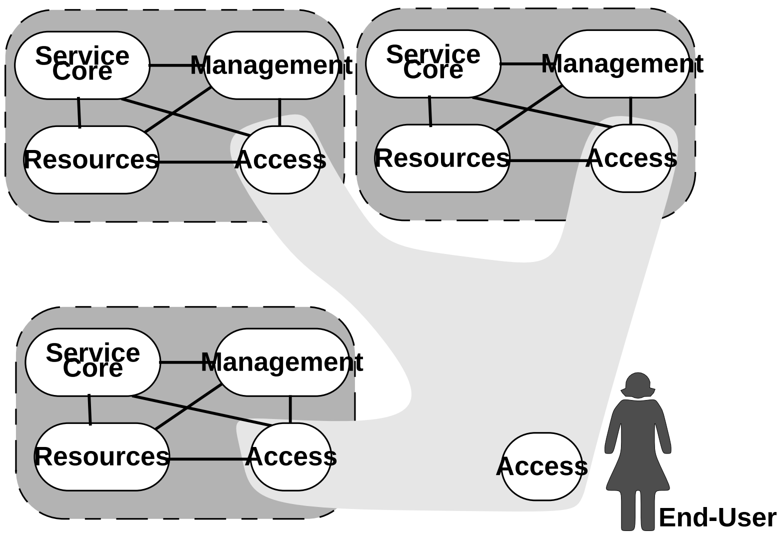


Figure 5-3. Generic access model

5.3 Separations from a management perspective

Management systems, as defined by TMN and OSI (Open Systems Interconnection) [21][22], define two types of separations: layering and functional separations. Layering, following a business, service, network, and element separation was presented above. Functional separations state that management systems can be divided into five broad areas of functionality, namely fault, configuration, accounting, performance, and security (FCAPS).

The definition of these will be expanded upon in Section 8. These functional separations can be applied to both the services and resources layer, wherever management functionality is exists in those layers.

5.4 Design separations

According to the computing architecture, a service will be defined as a collection of objects. As described above, objects comprising a service can be categorized into access, core, management and resource objects, where management objects can further be classified according to the FCAPS separations. It can be observed that these separations could also be suitable for other types of components that comprise the system. By component it is meant any object, or composition thereof, that is a unit of construction and reuse.

Taking all these separation principles into account during the design of TINA services and components can be a complex and confusing task. To provide usable design guidelines these separation principles have been rationalized into a simplified model called the *Universal Service Component Model* (USCM).

The USCM provides a classification scheme for the components of TINA compliant systems and services. Essentially, all services are modelled as consisting of a core surrounded by an access layer. The USCM provides a model for the interactions between components and between services. It is also the basis for the ability to construct new services from existing services through the recursive use of the USCM, although this has not been fully studied in TINA.

An object representation of a service may represent a large scale, complex service or a small, simple service. Whatever the size or complexity of the service, its structural organization should be consistent with the USCM division and is therefore the same as other TINA compliant services. This common service format is specified to promote consistency, reuse, and simplification of management.

The USCM is an abstraction and simplification of the separation principles described in this section. It provides both an internal and external description of the structure of any TINA service or component. The relationship between USCM components describe constraints on the group of related components that comprise the service.

5.4.1 The sectors of the access layer

The access layer of a service is divided according to three access aspects, viz usage, management and substance. Such a division yields Figure 5-4, which is a diagram that is commonly used to show a USCM model. Since the divisions of the access layer in this model imply divisions of the circular access layer, the divisions are referred to as *sectors*.

The three sectors are defined as follows:

- *Usage* sector: aspects in the usage sector provide interfaces for external components or services that motivate and directly control the operation of the service (e.g., the clients or users of the service). These components usually provide server interfaces to the service environment. Entities in the environment

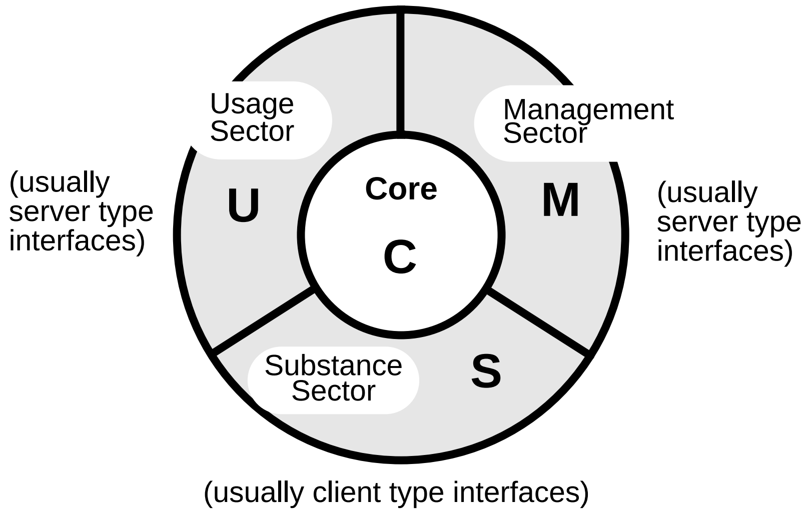

Usage Sector
Management Sector
(usually server type interfaces)
U
Core
C
M
(usually server type interfaces)
Substance Sector
S
(usually client type interfaces)

Figure 5-4. Primary structure of the Universal Service Components Model

see usage sector aspects as (server) interfaces to the TINA service. Components in the core see usage sector components as a normalized views of an external user of the core.

- *Management* sector: aspects in the management sector provide the logic and data to control the initialization, configuration, accounting, and other management and operational functions needed to establish, configure, and characterize the component itself. These aspects usually provide server or managed object interfaces to a managing system. Management sector aspects provide management interfaces to environment components and other services. These management interfaces are consistent with managed object models defined in the management architecture. The management aspects will control and manage all the sectors as well as core of the component. *Note that entities in the management sector only manage things within the same component and do not manage things in other components. For example a managed object in the management sector may only manage entities in the sectors of the same component.*
- *Substance* sector: aspects in the substance provide the internal representation of the external components that the component uses. The substance aspects commonly present user or client type interfaces to other services and resources in the service environment. Aspects in the core see substance sector aspects as a normalized view of an external resource for use by the core.

5.4.2 The service core

While these sectors are very general, they identify a sufficient set of support functions to allow description of all TINA services, objects and systems. The model is complete when the general access layer is combined with a core functionality that is unique for each component.

The *core* is the logic and data that defines the intrinsic operation, or *raison d'être*, of the component. The intrinsic operation controls the primary operation of the component without regard for the details of the supporting environment.

5.4.3 Use of USCM

The USCM can be seen to provide a high level checklist of what must be included to make a design complete. It can also be viewed as a model that upholds the various separation principles, which should lead to well-formed and reusable components.

The intention is for the USCM to be part of the overall architecture and thus to be used when defining any TINA software system and sub-systems, such as services, and management applications. However, until its usefulness is proved, its usage in specifying and designing TINA systems is optional.

6. Network architecture concepts

The purpose of the network architecture is to provide a set of generic concepts that describe transport networks in a technology independent way, and to provide mechanisms of the establishment, modification, and release of network connections. The network architecture defines a set of abstractions that the resources layer can work with. At one end it provides a high level view of network connections to services. At the other end it provides a generic descriptions of elements, which can be specialized to particular technologies and products.

6.1 Network layering and partitioning

The network architecture has been defined by taking into account principles of ITU-T Recommendation G.803 [19], and ITU-T M3100 [20]. G.803 describes the functional and structural architecture of SDH transport networks. However, many of the G.803 concepts are also applicable to networks other than SDH, such as ATM.

The major concepts of G.803 are partitioning and layering.

Partitioning means that a network may be decomposed into subnetworks and links[1] between them. Each subnetwork may be further decomposed into smaller subnetworks interconnected by links until the desired level of detail is reached. The lowest level of decomposition will usually be when a subnetwork is equivalent to a single network element (switch or digital cross-connect).

Transport networks can be viewed as composed of layer networks. Each layer network represents a set of compatible inputs and outputs that may be interconnected and characterized by the information that is transported. The inputs and outputs may be regarded as access points on the layer network. Characteristic information is defined as a signal of characteristic rate, coding and format. Generally, a layer network is closely tied to a specific type of network transmission and/or switching technology, e.g., SDH/SONET VC-4, ATM virtual channel (ATM VC) or ATM virtual path (ATM VP).

Layer networks may have client/server relations with each other. A link (between two subnetworks) in a client layer network is supported by an trail (end-to-end connection) in the server layer network. In the lowest layer network, a link will be a physical item (e.g. fibre, radio connection).

Figure 6-1 illustrates these concepts. It shows a network built of three layers. The lowest two layers are shown as being partitioned into sub-networks and links, with a client/server relationship between them.

1. TINA has modified some of the terminology from G.803, to take into account other network technologies, and other network models, such as that defined by M.3100.

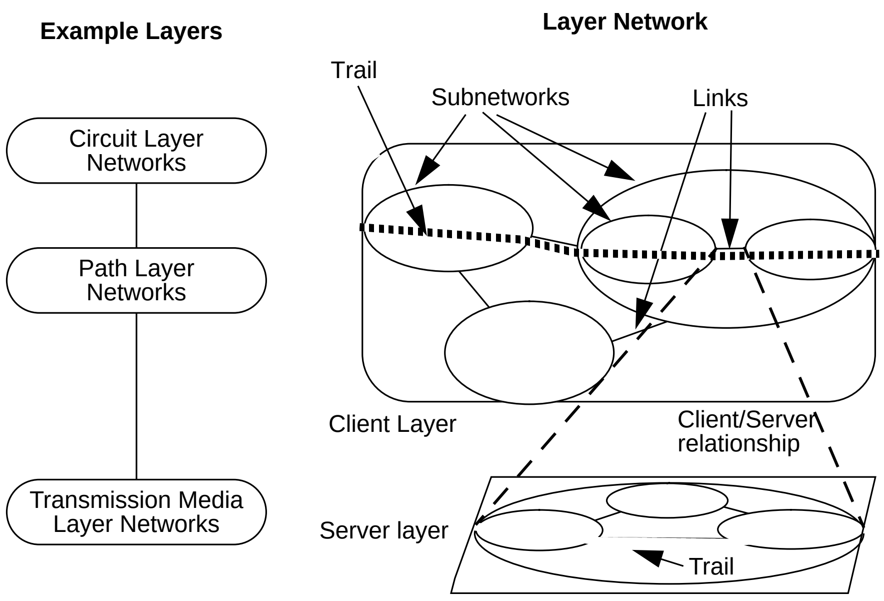


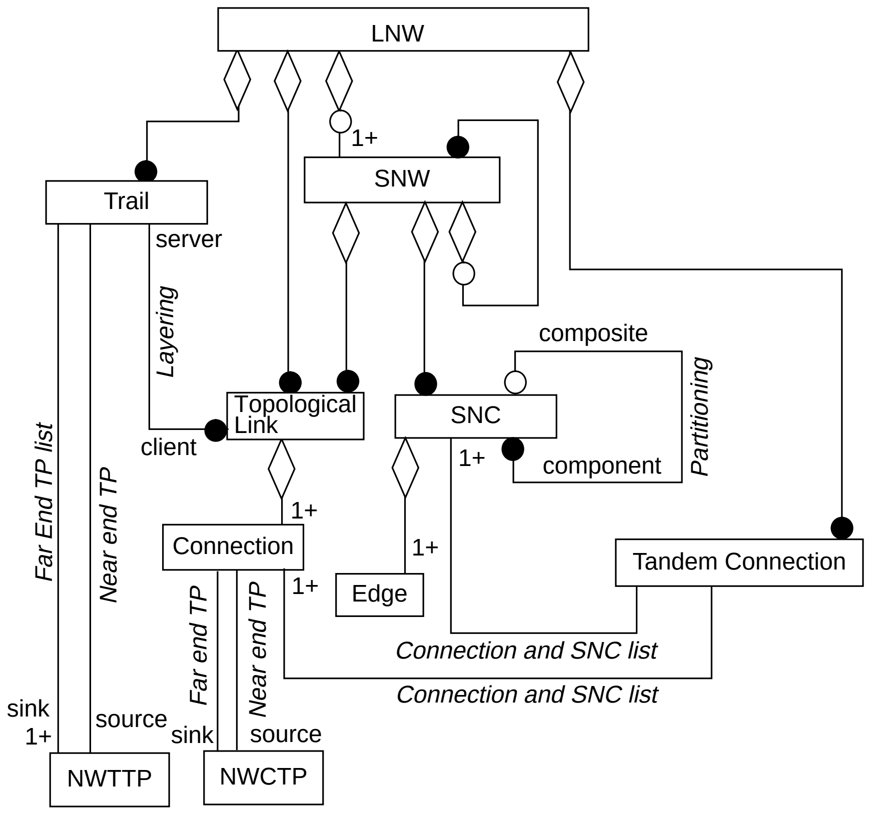
Figure 6-1. Partitioning, layering, and client/server relationships in transport networks

6.2 Network resource information model

The Network Resource Information Model (NRIM) is an information specification of transmission and switch technologies, based on the principles outlined above [8]. In this model technology dependant aspects have been extracted (e.g. remove differences between ATM and SDH switches). The model concerns how individual elements are related, topologically interconnected, and configured to provide and maintain end-to-end connectivity. The model therefore defines technology independent concepts that can be used to derive technology independent control and management functions. When designing and implementing a real network the technology dependant aspects must be taken into account as specializations of the generic model. The concepts found in this model include Layer Network, Sub-Network Connection, link connection, topological link, and network termination points. Figure 6-2 shows an OMT diagram of a fragment of the NRIM.

6.3 Connection graphs

The NRIM can be used to describe a detailed network. However it contains details that a user may not want to be, or should not be, aware of. Any use of a network, aside from management activity, must be as a result of using a service. It is important therefore to provide a service-oriented view of connectivity. The concept of connection graph is used for this purpose. A connection graph is an information specification containing vertices with ports, where ports are connected by lines and branches to represent connectivity (Figure 6-3).



Key:

SNC Subnetwork Connection
LNW Layer Network
SNW Subnetwork
NWTTP Network Trail Termination Point
NWCTP Network Connection Termination Point

Figure 6-2. A fragment of the NRIM

There are two basic types of connection graph. In a **logical connection graph**, the vertices represent computational objects, the ports represent stream interfaces, and the lines represent streams. In a **physical connection graph**, the vertices represent the computing nodes that the computational objects reside on, the ports represent network access points, and the lines represent the network connection portions of streams.

6.4 Connection management

The computational model that provides for the establishment, modification, and release of connections is called connection management. Connection management consists of a set of components that reside in the resources layer [9]. The NRIM and connection graph information specifications have been used to derive these components.

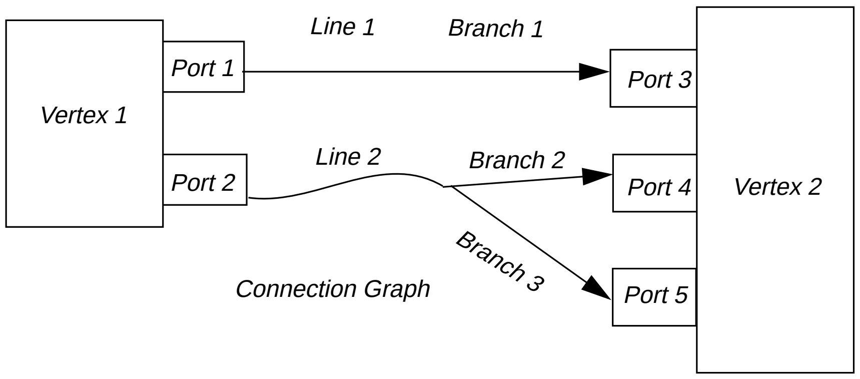


Figure 6-3. Connection graph

The **communication session manager** (CSM) provides an interface to service software. This interface consists of operations to build and modify logical connection graphs, such as to add, modify, and delete vertices, ports, and lines. Once a logical connection graph is built, the CSM translates it into a physical connection graph. The primary task in this translation is to identify the computing nodes that the computational objects reside on and the network access points to be used. The CSM will interact with a nodal connection manager on each node to request intra-nodal bindings. An intra-nodal binding is an association between a stream interface and a network access point (an example would be a socket in the UNIX operating system [29].) For each network connection required, the CSM will interact with a **connection coordinator** to establish connections between network access points. The CSM is responsible for performing the modification (e.g. bandwidth changes) and release requests from services. It is also responsible for reporting to services any changes in the underlying set of connections that affect a logical connection graph.

A typical transport network is composed of various layer networks each with their own characteristics. Each of these layer networks is controlled by a **layer network coordinator** (LNC) and a number of **connection performers** (CPs). A LNC is responsible for providing end-to-end connections through the layer network it is controlling. Each sub-network of a layer network is assigned a connection performer. A connection performer is responsible for establishing connections across a sub-network. The LNC and the CPs are specialized for the particular layer network they control.

A connection coordinator is responsible for carrying out the connection requests coming from the CSM (Figure 6-4.) A connection coordinator is not associated with a particular layer network, and it hides the various layer networks from the CSM. A connection coordinator will get a request to connect two or more network access points, providing a certain bandwidth and a certain quality of service. Based on this information, a connection coordinator determines which layer network to use, and requests the layer network coordinator to setup connections in that layer network. A layer network coordinator, in turn requests connection performers to establish sub-network connections. Where a sub-network is further partitioned, the connection performer will request the connection performers of the contained sub-networks to establish connections. This recursion continues until a connection performer that is responsible for a switch or cross-connect is reached. The connection performer in this case will interact will the element software acting as a proxy for that device.

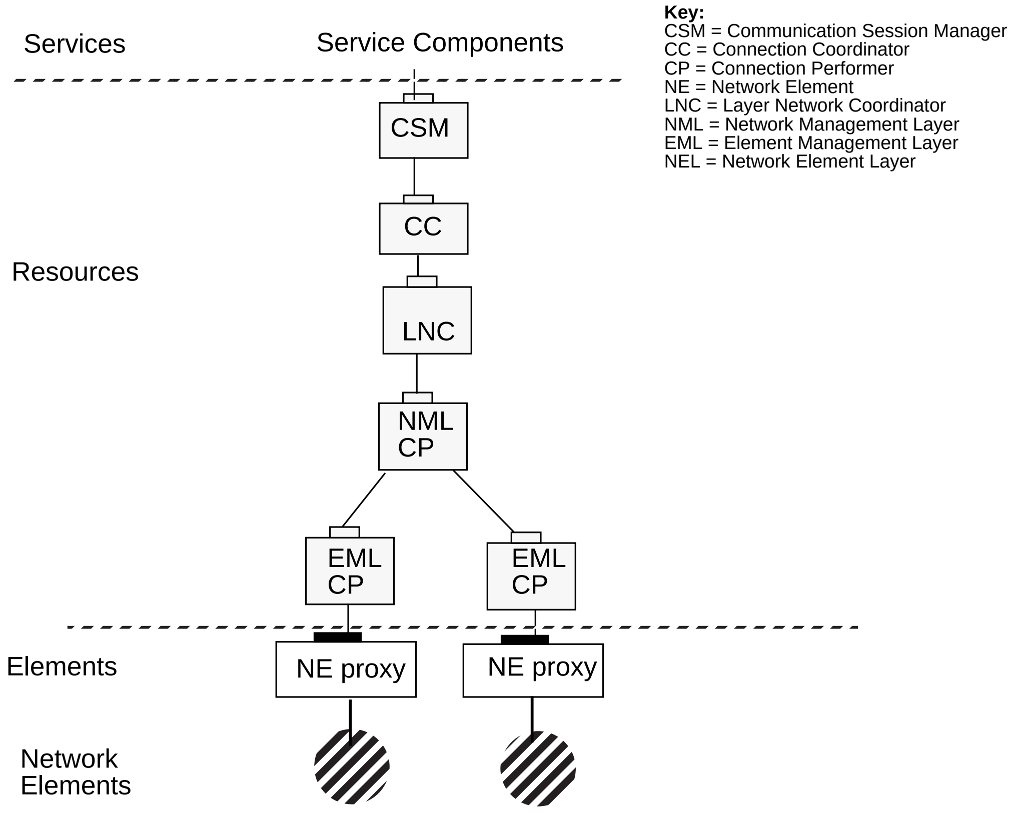

Key:
CSM = Communication Session Manager
CC = Connection Coordinator
CP = Connection Performer
NE = Network Element
LNC = Layer Network Coordinator
NML = Network Management Layer
EML = Element Management Layer
NEL = Network Element Layer
Services
Service Components
CSM
CC
Resources
LNC
NML CP
EML CP
EML CP
Elements
NE proxy
NE proxy
Network Elements

Figure 6-4. Connection management components

7. Service architecture

The service architecture aims to define a set of concepts and principles for the design, implementations, usage, and operation of telecommunications services [6]. It further aims to define a set of reusable components from which to build telecommunications services. Services and their environment are modelled as objects specified according to the three sets of modelling concepts in the computing architecture. There are three main sets of concepts and principles in the service architecture:

- **Session** concepts, which address service activities and temporal relationships,
- **Access** concepts, which address user and terminal associations with networks and services, and
- **Management** concepts, which address service management issues.

This section presents an overview of the session and access concepts. Service management concepts are presented in Section 8.3.1, under the management architecture.

7.1 Session concepts

Although services by their nature are different from each other, they all have a fundamental property in that they provide a context for relating activities. Such a context is termed a Session. As a generic definition, the term session represents a temporal period during which activities are carried out with the purpose of achieving a goal. Four types of sessions have been identified: **service session**, **user session**, **communications session**, and **access session**. Service sessions represent a single activation of a service. User sessions represent a single user's interaction with a service session. Communications sessions represent the connections associated with a service session. Access sessions represent a user's attachment to a system and their involvement in services.

A service session is the single activation of a service. It relates the users of the service together so that they can interact with each other and share entities, such as documents or blackboards. A service session contains the service logic. A service session is computationally represented by a **service session manager**. A service session manager offers two types of operational interfaces. The first is a generic session control interface. This provides operations that allow users to join and leave a service session. For certain services it may also offer operations to suspend and resume involvement in a service. The second type of interface will provide service specific operations, and will be dictated by the capabilities offered by the service logic.

The ability to suspend and resume involvement in a service is a desirable feature for some services. For example, consider a multi-media conference that occurs over several days. During the night, when the conference is not in use, it should be possible to release expensive communications resources. The service session can maintain state about the conference, such as the users and resources involved. Maintenance of state and the ability to suspend and resume involvement would avoid the need for tearing down and recreating the service each day.

A user session maintains state about a user's activities and the resources allocated for their involvement in a service session. Examples of state held in a user session include the user's accumulated charge, suspension and resumption history, and service specific state, such as the current page being edited in a distributed document editing service. When a user joins a service session, a user session is created. It is deleted when the user leaves. The service session maintains links to the user sessions and thus provides a group oriented view.

A communication session is a service-oriented abstraction of connections in the transport network. A communication session maintains state about the connections of a particular service session, such as the communication paths, end-points and quality of service characteristic. A communication session is only required when streams between computational objects are required. Computationally a communication session manager provides the features of a communication session (Section 6.4.)

An access session maintains state about a user's attachment to a system and their involvement in services. A user can attach to a system in order to launch or join service sessions. A user may be involved in many services at the same time, and an access session maintains state about this involvement. Computational aspects of access session are presented below.

The purpose of these session concepts is to separate out different concerns (see Section 5.2) and to promote distribution of functionality. The separation of access and service sessions allows for both the access methods and technology for different users to vary, and for the location of the users accessing the service to change whilst a service is in progression. The separation of service and user sessions allows for the distribution of functions and state, whereby the user session provides a local view, and the service session provides a collected view. This separation also supports the suspension and resumption of service involvement. The separation of service session and communication session supports the division of activities of the service from the set of connections that exist. There are two essential reasons for this split. Firstly, not all services will require the use of the transport network. In these cases, services will be provided through operational interfaces, the communication to which is supported by the DPE. Secondly, even if a service establishes connections on the transport network (i.e. has communications sessions), other activities may be taking place, and other users may be involved; there is not necessarily a one-to-one correspondence between those that are involved in a service and those that have transport connections as part of the service. An example would be three users cooperating on editing the same document, where a service session exists to coordinate the editing, and two of the users have an audio connection because they want to discuss between them some changes.

There are two important relationships that have yet to be addressed by TINA. The first is sharing of communication sessions between two service session. Currently it is assumed that a communication session, and the underlying connections, belong to a single session. A user involved in more than one session will have to have different connections. It may be desirable to share communication session between service sessions, especially if the same users are involved. The second relationship to be studied is the interaction between different service session. It is a goal of TINA to allow services to be built out of other services. However, work is required to study the implications of allowing service sessions to be constructed out of other service sessions.

The TINA architecture only defines generic service session concepts, principles, and components. No service specific aspects, such as certain service logics, are defined, although examples to demonstrate the concepts have been made.

7.2 Access concepts

Users need to have flexible access to services, in terms of the locations from which they access the service and the types of terminal they use. User access is therefore distinguished from terminal access. An agent concept is used in defining the access model. An agent is a computational object, or collection of objects, that acts on the behalf of another entity. Two types of agent have been identified: **user agent** and **terminal agent**.

A user agent is a computational object that represents and acts on the behalf of a user. It receives requests from users to establish service sessions, or to join existing service sessions, and creates[1] or negotiates with existing service sessions as appropriate. The creation of a service session by a user agent is subject to subscription and authentication checks. A user agent also receives and processes requests to join a service session from service sessions themselves. This is a form of in-coming call processing where another user has created a service session and invites the user to join in. User agents know the subscribed services that a user may create. This list can be presented to the user when the user logs onto his user agent. Comparing to current networks, the user agent is a place where service related signalling messages are received and processed.

A terminal agent is a computational object responsible for representing a terminal. It is responsible for obtaining the precise location of a terminal. Two examples are, which network access point a portable computer is attached to, and which cell a mobile phone is currently in.

In order to access a service, users must associate their user agents with terminal agents. This may form part of a logging on process to establish an access session. A user may be simultaneously associated with many terminals. For example, in a video conference a user may be using both a workstation and a telephone. Similarly a terminal may be simultaneously associated with many users, for example, when in a meeting all users associate their user agents with the telephone in the meeting room. On incoming session requests, a user agent has to determine which terminal agent should be contacted. If the user is currently accessing the system then an announcement could be issued to one of the terminals being used, and the user can instruct the user agent which terminal to use. Otherwise the user agent will have to determine which terminal and pass a request to the terminal agent who can alert the user. This determination can be done through user registration, prefer-

1. Creation is actually carried out by the user agent sending a request to a factory. A factory is a DPE service that can dynamically create objects.

ences, and defaults. An example would be a user that registers with his user agent the following location policy: office phone between 9am and 5pm, Monday to Friday; home phone any other time; voice mail service if either fail to pickup.

The power of the user and the terminal agent concept is that users and terminals can be located by finding the related agent. This can be achieved by using the trading services provided by the DPE. This avoids having to build in location knowledge and thus allows users and terminals to move around the network. User and terminal agents may also move around the system. This movement can be made transparent by the DPE, by updating entries maintained by the trading services for example. User and terminal agents should have high reliability and availability properties. These are required so that the system can rely on a contact point for locating users and terminals in an environment were both may be moving around.

7.3 Example

Figure 7.1 depicts a computational view of the access and the session concepts together. The shaded boxes of user and terminal agent represent service independent objects, and the white boxes depict service specific ones.

Assume that a user wishes to engage in a document editing session with another user. First, the user selects a terminal on which he will access the network. Assume it is a workstation with windows capabilities. As part of a login procedure the terminal agent and user agent are found - using the trading services of the DPE - and associated with each other. The user is then presented with a menu of capabilities. He selects the document editing option. A request is passed to the user agent to establish a document editing service session. The user agent creates a document editing service session manager, and joins the user to the session, causing a user session to be created. Another menu is presented to the user requesting the identification of the user to be called. On input of data, the user agent requests to the service session manager to join the new user into the session. The service session manager uses the identity to locate - again via trading - the requested user agent, and a request to join is passed over to this user agent. The remote user agent then alerts the terminal agent (after having determined which terminal to use) of the incoming session request. The terminal agent then alerts the terminal, by presenting a window on the user’s terminal (for simplicity assume the user was already logged on). The destination user accepts the request and the response is fed back to the service session manager. On seeing the acceptance, the service session manager creates a user session for the new user. The first user then requests the service session manager to set-up an audio-visual connection to the remote user, so that the two users can discuss edits to be made before carrying them out. The service session manager requests the communication session manager to establish a stream between end-user applications (computational objects) residing on the users’ terminals (the interface identities of which were passed over in an earlier request or response). For simplicity the details of the connections are not shown in the figure nor further discussed here. When the stream is established, a response is passed back to the originating user. The two parties can then engage in an audio-video conversation, where they discuss the changes to be made to the document.

The originating user requests the service session manager to open the document, and the document appears on the users' screens. Commands to edit the document are sent to the service session manager, and the changes are reflected back on the users' screens.

At any time either party may leave the session. This will result in the other user being notified and the deletion of the service session. During the session, subject to permission, each user may request another user to join.

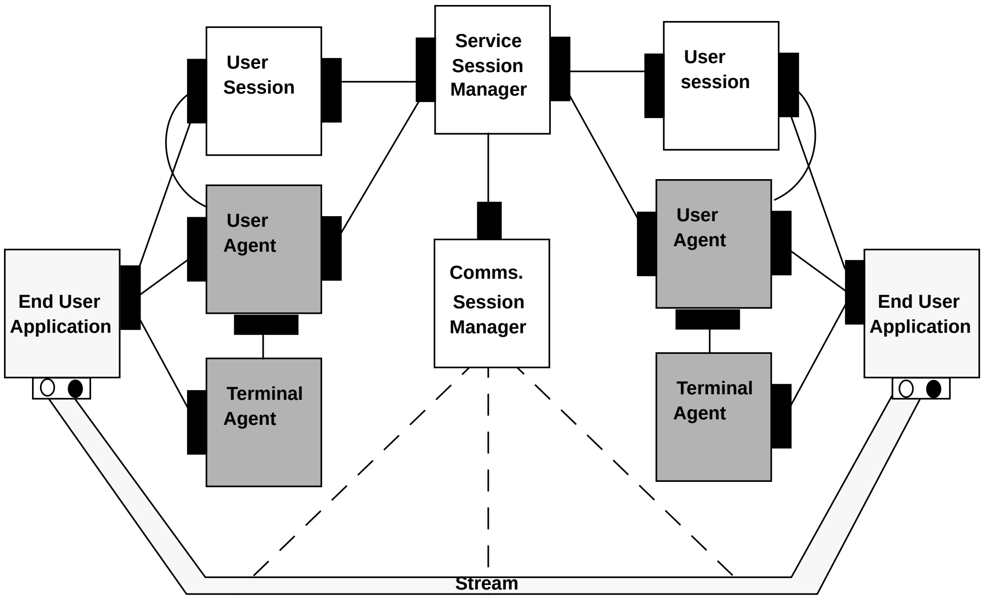


Figure 7.1: Access and session components

8. Management architecture

The TINA management architecture provides the concepts and principles to build management systems that can manage the entities in TINA systems. As with the service architecture, the computing architecture is used to define object types and interfaces that should be used to manage TINA services, resources and infrastructures. This section presents only general management concepts and principles in TINA. Details can be found in [7] and [6].

8.1 Types of management

As presented in Section 5, a TINA system consists of a computing environment upon which service, resources, and element applications run. This gives rise to two basic types of management.

Computing management[1] involves the management of the computers (NCCE), DPE, and of the software (in general terms) that runs on the DPE. Management here does not concern itself with what applications are doing nor on application specific management. The main concern is the deployment, installation, and operation of software and computing nodes.

Telecommunications management involves the management of the transport network, the management of the applications that use and control this network, and the management of services.

These two types of management are very broad, and are themselves broken down into subtypes of management. Telecommunications management is broken down into service, resource, and element management, in much the same way as TMN systems. Computing management is divided into generic software management, such as deployment, configuration, and instantiation of software, and management of the DPE and computer environments. Thus there is a relationship between the service, network and computing architectures and management concepts and principles.

8.2 Generic management

Even though different types of management have been identified, the TINA architecture defines a set of generic management concepts and principles.There are two sets of concepts and principles for generic management:

- Functional separations, that break down the problem of management into distinct areas of concern.
- Modelling of management systems, that defines how to express management relationships and operations.

1. The term computing management is used in preference to the term systems management. The two terms approximately cover the same areas of concern. However, it is felt that computing management is more meaningful in the TINA context, and avoids the confusion that arises though use of terms such as telecommunications systems and computing systems.

The generic management concepts and principles recognize that there are good ways to manage entities irrespective of what those entities are. Therefore, these principles should be applied to telecommunications and computing management. Applying the same generic concepts should lead to more consistent and easier to understand management systems, and maximizes the ability to reuse support services and tools. Figure 8-1, depicts the two main classes of management and highlights that common concepts and principles should be applied to both.

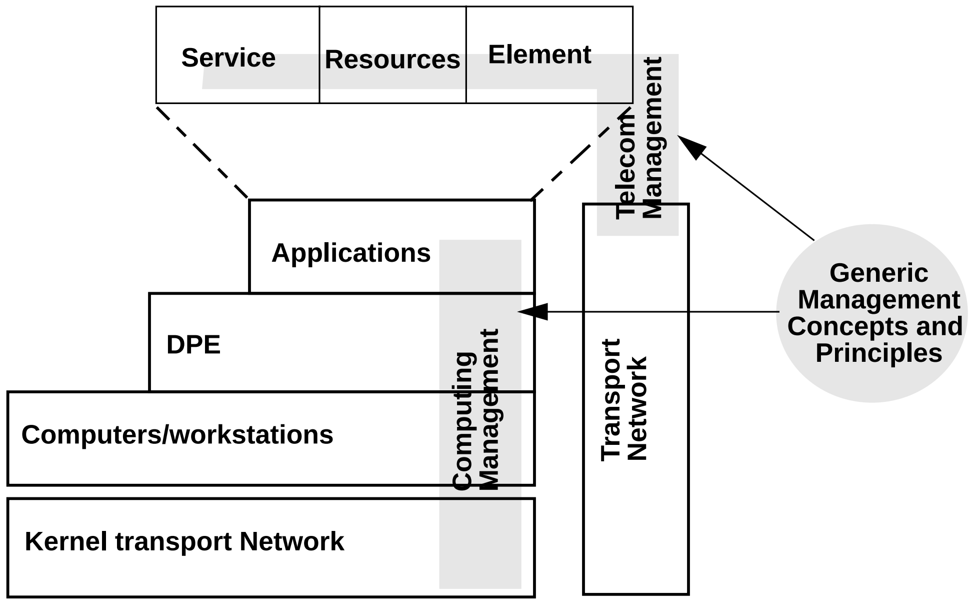


Figure 8-1. Two types of management

When applied to each area, the generic concepts and principles will be specialized (and may be extended) so as to be suitable to the particular type of entities being managed.

8.2.1 Functional separations

Management can be divided into five aspects: **fault**, **configuration**, **accounting**, **performance** and **security**. This is the FCAPS separations found in OSI systems management. All five aspects must be included if a system is to be fully manageable. Each of these aspects define generic concepts and principles. For example, accounting management defines concepts such as accountable object, usage data and account metering, and provides principles on how these concepts should be used to account for any set of entities. A generic accounting management model is independent from the types of entities being accounted for. It does not restrict accounting to that of resource usage. The accounting model has to be specialized and/or extended for the service, network, and computing architectures of TINA. For example, in the service architecture, the generic accounting model is extended with concepts of charging and billing.

The five areas of management are each very broad in scope. They are basic guidelines to divisions of functionality. When applying these areas to particular management problems further subdivision may be required. This may result in specific models that cannot be generalized. There are two major refinements in TINA. In the service architecture, subscription management is considered a functional area that is part of the broader aspects of configuration management. Since subscription management occurs only at the service layer there is no need for a generic model to cover different layers. In the network architecture, connection management, which applies management principles for the real-time establishment, modification, and release of transport network connections, is considered a functional area that is part of the more broader aspects of configuration management.

8.2.2 Modelling management systems

Management in TINA systems is achieved by the construction of models consistent with the computing architecture. In the computational viewpoint, managed entities are represented as objects and provide operational interfaces that allow managing objects to manipulate them. There may be more than one managed entity represented by a computational object. In these cases, the interface provides managing systems with a way to manipulate individual and groups of managed entities. Where a computational object represents more than one managed entity, the computational object may have to perform filtering and scoping in much the same way as an agent does in OSI systems management. Filtering and scoping are a set of functions that decide which entities should be affected by a management operation. Managing objects may themselves by managed, and thus may also provide interfaces for other objects to manage them.

Management services are applications that provide users with the ability to observe and control the activities of a system. In TINA, management services can relate to telecommunication, or computing management, or a combination thereof. Examples are a subscription application, that provides electronic forms for the entry of customer details, a traffic management application, that provides graphic displays of current bottlenecks in the transport network, and a software deployment application, that graphically depicts nodes and allows the user to drag and drop software modules onto nodes.

Management services should be designed according to the service architecture. This means that a user wishing to use a management service will, via his user agent, establish an appropriate service session. The logic of the service session will provide the management capabilities, for example, the generation, receipt, and processing of electronic forms for a subscription application. The use of the same architecture for user applications, irrespective of its function, allows for common generic components to be defined and provides greater consistency in the overall system.

Management services may be provided to customer organisations as well as service providers and network operators. An example management service for a customer is an application that allows a customer administrator to manipulate a private numbering plan for a virtual private network.

It is worth noting that management services may themselves need managing. This is called second order management. This is made explicit in the USCM, where the core of a management component manages other entities, and the management sector contains capabilities to manage the component itself. The spilt between computing and telecommunications management also aids this problem, in that telecommunications management applications may be managed (in generic terms) by computing management applications. However, the proper termination of this recursion has not yet been studied in TINA.

8.3 Telecommunications management

8.3.1 Service management

The Service Architecture uses the generic management functional separations defined above, and extends and specializes them to be suitable for service management.

From the service perspective, fault management relates to restoration of a service to subscribers within the time agreed in their respective contracts. Tracking of progress on restoration of the fault for the service provider and subscriber is facilitated by use of Trouble Ticketing. Service fault management has yet to be addressed in TINA.

Configuration management in the service architecture can be divided into two main subsets: service life-cycle management and customer life-cycle management. Service life-cycle management deals with the deployment, maintenance and withdrawal of services. These activities will rely heavily on software configuration management, described below. Service life-cycle management has yet to be adequately addressed in TINA.

Customer life-cycle management includes those activities which are necessary before, during, and after subscription. Pre-subscription activities include promotion of the service, negotiation with the (potential) customer and formulation of the subscriber contract. Post-subscription activities includes dealing with enquiries, and monitoring of customer satisfaction. Subscription Management aspects are defined by considering the Service Provider, Customer (Subscriber) and End-user stakeholder roles. The service provider will offer a range of capabilities for a service which may be tailored to meet the requirements of a customer. The customer will select those service capabilities from the service offering which will meet his requirements. These will be agreed between the customer and service provider and form the basis of a contract. The customer will then offer these capabilities, or a restricted set of them, to the end-users. A subscription model is defined in [6].

Accounting management for services is primarily concerned with the application of tariffs to usage of services, deriving charges and billing for subscribers. The mechanisms for obtaining and storing usage information are defined in the generic accounting model. The billing periodicity will be explicit in the contract, as should procedures for non-payment and dispute. It should be possible in a TINA environment for customers to obtain their bill for themselves on demand. It is obvious here that security of information and security of the accounting management system is particularly significant. An account management model is defined in [6].

Performance Management is concerned with maintaining agreed Quality of Service (QoS) levels on services to customers. This is normally achieved by monitoring the performance of entities in the network. It is essentially a preventive activity in that performance degradation should be detected before a “hard” fault occurs resulting in loss of service(s). QoS will be characterized as part of the contract. Performance management is yet to be addressed by TINA.

Security management in the service architecture addresses who can access what services and from where. These issues are not yet fully addressed in TINA.

8.3.2 Network management

The Network Architecture uses the generic management functional separations defined above, and extends and specializes them to be suitable for network management. Since the network architecture covers different network and network element abstractions, network management in the network architecture encompasses both the network management layer and the element management layer of TMN.

Fault management in the network architecture encompasses detection, isolation, and correction of improper behaviour of network resources. Alarm surveillance permits monitoring of resources and makes information about fault status available outside the resource itself. Fault localization identifies the specific resources that are responsible for improper behavior within the network. Fault correction is concerned with the restoration of resources currently in a fault condition. Testing/Diagnostic is associated with analysis of circuits or equipment and reporting of the results. Trouble administration is an activity that enables troubles to be reported and their status tracked.

Configuration management in the network architecture is divided into two sub-areas: resource configuration and connection management. Resource configuration includes supporting the installation of network resources, provisioning of network resources to make them available for use, monitoring and control of resource status (e.g., available or not available for service use) It also includes the management of the relationships among the resources.

Connection management is concerned with the setup, maintenance and release of connections, including the specification of a connection model. Traditionally connection management is considered as control operations, which are viewed as being different from management. In TINA, these control operations are seen are real-time, or dynamic, management operations. Connection Management is used by service architecture components whenever a service requires connections. Section 6.4 presents an overview of connection management.

Accounting management in the network architecture is responsible for providing the ability to collect resource usage information and to apply the adequate charges for that usage. Metering and charging are the two major concerns. Metering is the task of recognizing and recording information relevant to the usage of a resource in a meaningful way. Metering information will be forwarded to service management applications. No specific network accounting model has yet been defined in TINA.

No concepts and principles have yet been defined for performance and security management of networks.

8.4 Computing management

Computing management is broken down into two areas. Software management considers how to install, create, delete and withdraw software. Infrastructure management considers how to manage the NCCEs, the DPE (kernel and services), and the kernel transport network. The direct management of NCCEs is outside the scope of TINA, although the DPE may provide facilities that can be used to indirectly manage the underlying NCCEs.

The work on computing management is immature. However, to provide an insight into how it is being approached, the use of the layering principles defined in Section 5.1 is described.

For computing management, the entities to be managed will be represented as objects in the element layer. In some cases these objects will be proxies for the entities being managed, in other cases these objects may be the actual entities subject to management. An example of the first case is a loadable software module. The module itself is not an object, but will be represented as an object in the element layer for computing management purposes. An example of the second type is a computational object that offers an operational interface with operations to suspend and resume execution. In this case there is no need to provide a proxy, as a managing system can directly invoke operations on the object (element) itself.

The resources layer should maintain a list of elements and the relationships between them. For software management, the resources layer should maintain knowledge of what software modules are installed on what computing nodes, and on what modules comprise an application. The latter is required when an operation needs to be carried out on all the module of an application, such as to remove an application. For infrastructure management, the resources layer should maintain knowledge of the computing nodes in the system, what their capabilities are (such as the presence of DPE services), and how they are connected to each other through KTN resources.

The service layer will contain objects that provide for computing management services, that can be used by administrators. Examples are software deployment services, that interact with the resources layer to add new applications and software modules into the system, and performance monitoring services, that interact with the resources layer to monitor execution loads on the computing nodes and traffic flows between the nodes.

9. Design guidelines

This section presents an overview of a methodology that can be used to apply the TINA architecture to the design of services and service components for TINA systems[1]. In this section, only an overall description of a development process is presented. Further details can be found in [11]. Also presented in this section is an overview of some tools that have been developed to support the use of notations. The notations and tools have not been integrated into the methodology.

Use of the methodology, notations and tools discussed here are internal principles, and their use is not required outside the Core Team. Any methodology, notation, or tool can be used provided the resulting systems conforms to the TINA architecture.

9.1 Service development methodology

The service development methodology addresses the needs of TINA service developers working in various service and system environments to use a common approach to service development. The objective of such a common approach is to foster development of cooperative and coordinated products that can be used as components for building services. These products will go beyond any list of component types that might be generated as part of the initial TINA architecture. With such an open-ended product set as a base, the objective of building services from preexisting services and components is feasible. If such a coordinated product set does not exist, TINA service developers will be limited to building from primary components or capabilities defined as part of TINA itself. This situation would be similar to building from a “capability set” and functional entity system such as found in the IN architecture. This idea is a cornerstone of the TINA approach to service development, namely, that services are developed from an evolving environment of components and each product can build on the success of preceding products rather than starting from the same component set each time. This gives the TINA model great evolutionary flexibility, but it also presents some serious challenges when mapping a creation process.

The basic paradigm of the service development methodology is the Description Plane Model (DPM). The DPM defines five main phases, or planes, of a development process, where each plane is devoted to certain development activities. The main motivation for this model is the need to relate the viewpoints of the computing architecture together from a methodological standpoint.

The five planes of the DPM are objectives, definition, design, implementation, and physical. Figure 9-1 shows these planes and the relationships between them.

Typically, a development cycle will begin in the *objectives plane* with the creation of a description of the service in terms of requirements, and stakeholder models. Stakeholder models are used to identify the roles that people and organizations will be carrying out with respect to a service and/or its support environment, and should be used to analyze expectations. An enterprise model description of a service will be the output of the objective plane.

1. The methodology described here reflects work up to December 1994, and may not reflect the current version of the methodology.

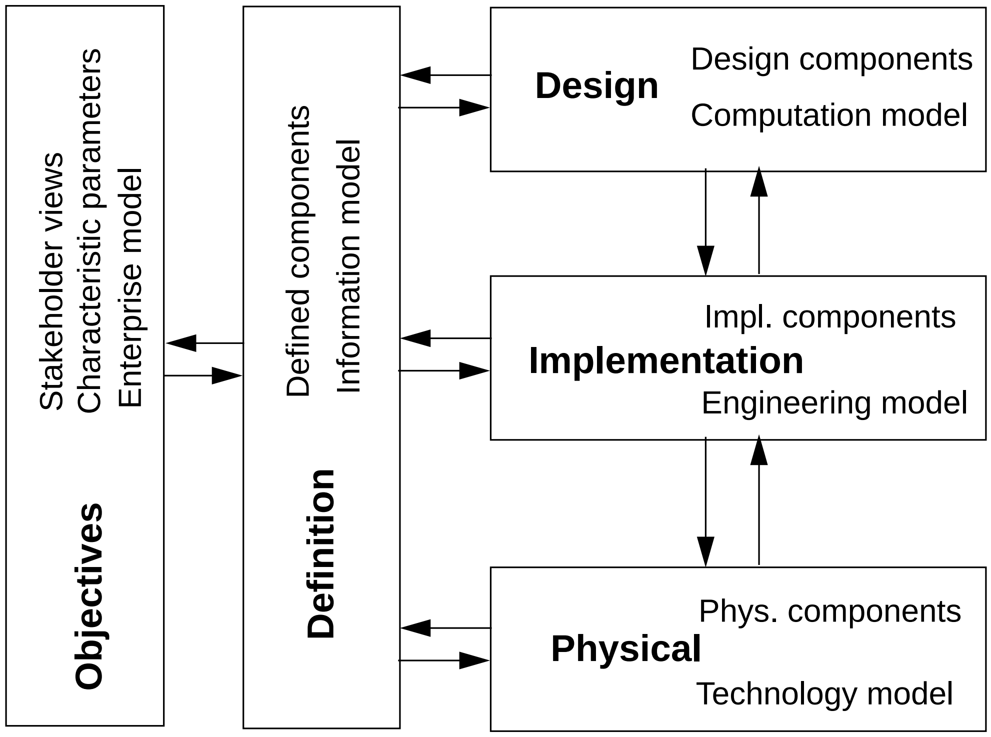


Figure 9-1. Description Plane Model

Typically, the second step of service development takes place in the *definition plane*. In the definition plane, the enterprise model is used as input to construct an object-oriented description of the service and its environment, with the emphasis describing *what* the service is, and not on how it should be implemented. The output from the description plane is an information model that reflects the details of the “problem space” of the service. The “problem space” information model describes the world of the service in terms of all of the pertinent actors (derived from stakeholders) and components. The information model will contain descriptions of the environment of the service as well as a description of the components used to build the service and their relationships, and should be structured according to the USCM.

The third plane of the DPM is the *design plane*. The design plane contains object-oriented descriptions of the components of logic and data that will be used to actually implement the service. The output of the design plane will be a computational model that defines the units of programming (computational objects). The design plane therefore focuses on *how* to implement the problem domain from the definition plane. A clear requirement/satisfaction relationship must be specified between entities in the definition and design planes.

The fourth plane of the DPM is the *implementation plane*. The implementation plane contains descriptions of the deployable software modules that comprise the service. The implementation plane descriptions are concerned with issues such as distribution, failure mode operation, security, etc. The outputs of the implementation plane are an engineering model, and implementation code. The engineering model will include a deployment model that will express requirements and guidelines on how to deploy the software modules in a physical system. In order to produce the implementation code, knowledge of the technology that is to be used is essential (e.g. types of network and computing technology). Therefore a technology model should serve as an input. This may require the construction of a partial technology model that identifies the programming languages, operating systems, hardware, and communication protocols that will be used in the physical system.

The fifth plane of the DPM is the *physical plane*. The physical plane is concerned with the operational deployment and execution of the software modules. The output will be a combined technology model and physical deployment model. A technology model describes in detail the software and hardware technology, and the topology of the system. The physical deployment model shows which software modules are installed on what nodes.

Figure 9-1 shows the development process branching out from the definition phase. There are two main reasons for this. Firstly, the design and implementation of a service cannot be carried out until the requirements have been captured and a formal model of what the service is trying to achieve has been constructed. However, this does not imply that activities in the design, implementation, and physical planes cannot proceed until the complete definition is reached. A service, early on in its definition, can be broken down into major sub-systems, and the development of these sub-systems may proceed in parallel. Secondly, to support a multi-vendor environment for the construction of software and the interoperability of a service between different service provides requires agreements on what system is to achieve. Whilst TINA has not fully investigated conformance and interoperability, the basic requirement to support separate developments is an agreement on the (formal) definition of the service. The models developed in the definition plane provide this coordination.

9.2 Notations and tools

Even though the intent and scope of the Core Team is not to build a service creation environment (SCE), some tools are currently used and customized by the Core Team. They relate to the specification and the development phase of the life cycle model. One is an OMT tool (Software Through Pictures) that has been used and customized in TINA-C to model information objects [10]. From this modelling, quasi-GDMO-GRM specifications can be derived automatically as a text representation of the information model. The other tool is the ODL specification pre-processor that will produce code targeted towards CORBA platforms. A first version of this tool converts ODL specifications into OMG IDL specifications.

Figure 9-2 depicts the development support tools used in the Core Team. The solid arrows represent inputs and outputs to the tools. The dashed arrow represents informal (i.e. non-automated) input. It is anticipated in the future that a complete tool chain will be developed allowing for information models (OMT diagrams and/or q-GDMO specifications) to be input to a tool that could produce outline ODL specifications, and for the ODL compiler to produce outline code in selected programming languages. It is also envisaged that a tool for the graphical depiction of computational models will be developed. This may be through modifications to the OMT tool.

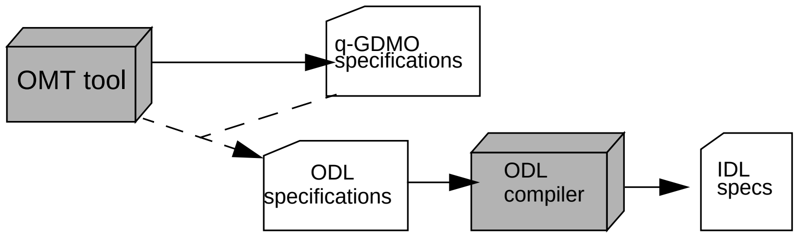


Figure 9-2. Design support tools

10.Acknowledgements

The authors wish to thank all those involved in the development of the architecture, namely all the TINA-C core team members. We also wish to thank the document reviewers (Paul Prozeller, Thomas LaPorta, Yann Lepetit, Gunnar Nilsson, Paul Vickers, Hiroshi Ishii, Javier Huelamo, Harm Mulder, Mike Schenk, Hendrik Berndt, Erik Colban, Dave Brown, and Rickard Jansson) for technical and practical suggestions.

Martin Chapman
BT
UK

Stefano Montesi
Telecom Italia
Italy

References

TINA-C Documentation

[1] *An Overview of the TINA Consortium Work Effort*, Document No. TB_G.HR.001_1.0_93, TINA-C, December 1993.

[2] *Requirements Upon TINA-C Architecture*, Document No. TB_MH.002_2.0_94, December 1994.

[3] *Information Modelling Concepts*, Document No. TB_EAC.001_3.0_94, TINA-C, December 1994.

[4] *Computational Modelling Concepts*, Document No. TB_NAT.002_3.1_94, TINA-C, December 1994.

[5] *Engineering Modelling Concepts (DPE Architecture)*, Document No. TB_NS.005_2.0_94, TINA-C, December 1994.

[6] *Definition of Service Architecture*, Document No. TB_MDC.012_2.0_94, TINA-C, December 1994.

[7] *Management Architecture*, Document No. TB_GN.010_2.0_94, TINA-C, December 1994

[8] *Network Resource Information Model Specification*, Document No. TB_LR.010_2.0_94, TINA-C, December 1994.

[9] *Connection Management Architecture*, Document No. TB_JJB.005_1.0_94, TINA-C, December 1994

[10] *STP Tool Documentation*, Document No. TB_EC.004_1.0_94, TINA-C, December 1994

[11] *TINA-C Service Development Methodology*, Document No. TP_DKB.010_0.1_94, TINA-C, December 1994.

External Documentation

Related Activities

[12] ANSA, *The ANSA Reference Manual Release 01.01*, Poseidon House, Castle Park: Cambridge, UK, July 1989.

[13] Bellcore SR-NWT-002282, *INA Cycle 1 Framework Architecture,* Issue 2, April 1993.

[14] Bellcore, SR-NWT-002268, *Cycle 1 Specification for Information Networking Architecture (INA)*, Issue 2, April 1993.

[15] Bellcore, TR-STS-000915, *The Bellcore OSCA Architecture*, Issue 2, October 1992.

[16] RACE Project R.1093 (ROSA) Deliverable 93/BTL/DNR/DS/A/005/b1, RACE*, The Rosa Architecture, Release Two*, Version 2, RACE, May 1992.

[17] Abimbola Oshisanwo, et. al., *The RACE Open Services Architecture project*, IBM Systems Journal, Vol. 31, No. 4, pp 691-710, 1992.

Relevant Standards

[18] ITU-T Recommendations on Intelligent Networks, Q.1200-Q.1290, approved 1992. In particular, CS1 Recommendations are in the Q.121x series.

[19] ITU-T Recommendation G.803, *Architectures of Transport Networks Based on the Synchronous Digital Hierarchy (SDH),* June 1992.

[20] ITU-T Recommendation M.3100, *Generic Network Information Model*, 1992.

[21] ITU-T Recommendation M.3010, *Principles for a Telecommunications Management Network,* 1993

[22] ISO/IEC DIS 10165-4 / ITU-T Recommendation X.722, *Information Technology - Open Systems Interconnection - Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects (GDMO),* International Organization for Standardization and International Electrotechnical Committee, September 1991.

[23] ISO/IEC 10165-7 / ITU-T Recommendation X.725, *Information Technology- Open Systems Interconnection - Structure of Management Information - Part 7: General Relationship Model,* International Organization for Standardization and International Electrotechnical Committee, January 1993.

[24] ISO/IEC JTC1/SC21 10746-1/ ITU-T Draft Recommendation X.901, “*Basic Reference Model of Open Distributed Processing - Part 1: Overview and Guide to Use*”, November 1992

[25] ISO/IEC JTC1/SC2110746-2.2/ITU-T Draft Recommendation X.902, “*Basic Reference Model of Open Distributed Processing - Part 2: Descriptive Model*”, November 1992

[26] ISO/IEC JTC1/SC2110746-3 /ITU-T Draft Recommendation X.903, “*Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model*”, November 1992

Object-Orientation

[27] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall: Englewood Cliffs, N.J.:, 1991.

[28] OMG Document Number 91.9.1, Draft, *The OMG Object Model*, September 1991.

Operating Systems

[29] Maurice J. Bach, *The Design of the UNIX*™ *Operating System*, Prentice Hall: Englewood Cliffs, N.J.:, 1986.

Glossary

- **Computing architecture**: A set of concepts and principles for designing and building distributed software and the software support environment.
- **Deployer/Withdrawer:** A deployer/withdrawer places and configures developed software and hardware modules, received from a developer, into a network, and confirms that the system runs correctly, and removes the modules when they are no longer needed.
- **Developer:** A developer implements a system and passes the implementation to the deployer.
- **Distribution Transparency:** The concept of hiding from applications details and complexities introduced by distribution.access transparency. The nature of transparency is classified into: transaction transparency, location transparency, failure transparency, federation transparency, migration transparency, group transparency, and resource transparency.
- **Management architecture**: A set of concepts and principles for the design, specification, and implementation of software systems that are used to manage services, resources, software, and underlying technology.
- **Network architecture:** A set of concepts and principles for the design, specification, implementation, and management of transport networks.
- **Network Provider:** A network provider provides transport and routing capabilities to both subscribers/users and service providers so that services may be accessed and operated respectively. A provided service will use the basic transport capability provided by the network provider, such as switching, transport, or lower layer protocol processing.
- **Overall Architecture**: A set of concepts and principles that embody the general features of TINA.
- **Service Architecture**: A set of concepts and principles for the design, specification, implementation, and management of telecommunication services.
- **Service Broker:** A service broker provides to a subscriber the service that is offered by another service provider.
- **Service/Network Designer:** A designer designs a telecommunications service/network according to service/network requirements. A designer produces specifications, and passes them to developers. In designing a service/network system, the designer should follow a set concepts and principles defined by the TINA-C architecture.
- **Service/Network Manager:** A manager manages the operation of services/networks to maintain their normal condition. A service manager negotiates with subscribers regarding their subscription and maintains and updates subscription data. Also a service manager, in collaboration with a network manager, calculates charges and sends bills to subscribers.
- **Service Provider:** A service provider provides services to subscribers/users, and charges subscribers for the services.

- **Software Architecture**: A set of concepts and principles for the structuring and provisioning of software.
- **Subscriber:** A subscriber purchases service capability provided by a service provider so that users can utilize this capability. The term **customer** is a synonym for subscriber.
- **User:** A user utilizes a telecommunications service, and also utilizes network capabilities provided by a network provider in order to access and utilize services. A user who interacts with a service to obtain the effect of the service is called an **end-user**. A user who interacts with a service for other reasons than to obtain the effect, such as to manage a service, may be assigned other roles, such as service manager.

Acronyms

- **ANSA:** Advanced Networked Systems Architecture
- **ATM:** Asynchronous Transfer Mode
- **B-ISDN:** Broad Band Integrated Services Digital Network
- **CC:** Connection Coordinator
- **CCITT:** Comite Consultatif International Telegraphique et Telephonique (cf. ITU-T)
- **CORBA:** Common Object Request Broker Architecture
- **CP:** Connection Performer
- **CPE:** Customer Premises Equipment
- **CPN**: Customer Premises Network
- **CSM**: Connection Session Manager.
- **DPE:** Distributed Processing Environment
- **DPM:** Description Plane Model
- **ETSI:** European Telecommunications Standards Institute
- **FCAPS:** Fault, Configuration, Accounting, Performance, and Security.
- **GDMO:** Guidelines for the Definition of Managed Objects
- **GRM:** General Relationship Model
- **IN:** Intelligent Network
- **INA:** Information Networking Architecture
- **ISDN:** Integrated Services Digital Network
- **ITU-T:** International Telecommunication Union - Telecommunication Standards Sector
- **KTN:** Kernel Transport Network
- **LNC:** Layer Network Coordinator
- **MO:** Managed Object
- **NE:** Network Element
- **N-ISDN:** Narrow Band Integrated Services Digital Network
- **ODP:** Open Distributed Processing
- **OMG:** Object Management Group
- **OMT**: Object Modeling Technique
- **OO**: Object-Oriented or Object-Orientation
- **OSF:** Open Software Foundation
- **OSI:** Open Systems Interconnection

- **POTS:** Plain Old Telephone Service
- **PSTN:** Public Switched Telephone Network
- **q-GDMO/GRM:** quasi GDMO/GRM
- **QoS:** Quality of Service
- **RACE:** R&D in Advanced Communications technologies in Europe
- **ROSA:** RACE Open Service Architecture
- **SCE:** Service Creation Environment
- **SDH:** Synchronous Digital Hierarchy
- **TINA:** Telecommunications Information Networking Architecture
- **TINA-C:** TINA Consortium
- **TMN:** Telecommunications Management Network
- **UNI:** User Network Interface
- **USCM:** Universal Service Component Model