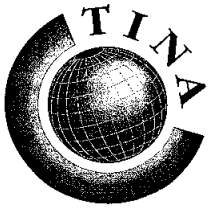


Telecommunications
Information
Networking
Architecture
Consortium

TINA-C Baseline

Issue Status: Final

Service Architecture

Annex

Version: 5.0

Date of Issue: 16 June, 1997

Abstract: This constitutes the Annex of the TINA Service Architecture document. It presents descriptive additional concepts and principles of the TINA service architecture, which is not to be considered as core TINA concepts.
It includes architecture refinements, intermediate results on various topics and guidelines on how to apply the architecture.

Main Authors:	**C. Abarca, P. Farley, J. Forslöw, J. C. García, T. Hamada, P. F. Hansen, S. Hogg, H. Kamata, L. Kristiansen, C. A. Licciardi, H.Mulder, E. Utsunomiya, M. Yates**
Editor:	**Lill Kristiansen**
Stream:	**Service Stream**
Workplan Task:	**Service Architecture**
File Location:	**/u/tinac/97/services/docs/sa/sa5.0/final/ also available through http://tinac.com:4070/97/services/www/sa50.html**

Table of Content of Annex

A-1 Introduction

This annex describes a set of useful concepts and principles related to the service architecture, that are part of the service architecture, but that are not a reference for compliance to TINA. In other words, a system does not need to adhere to any of the concepts presented in this annex in order to be TINA-compliant.

A-1.1 About this part of the document

The content of this part of the document is heterogeneous. More precisely, it contains:

- Concepts, principles, rules and guidelines that are descriptive. This material constitutes a refinement of the architecture with respect to what is defined in the main body. It does not represent a reference for compliance to the TINA service architecture; it is intended as a set of suggestion to designers to build a service architecture implementation.
- Clarifications on how particular goals can be achieved using the prescriptive part of the service architecture.
- Intermediate results on particular topics that not yet mature to be included in the prescriptive part of the architecture. These results identify directions for future work, both inside and outside of the Consortium, and therefore need to appear in the Service Architecture document. Future extensions to the main body of the document are likely to rely on these results.

Section A-1 (this chapter) provides an introduction to the annex part of the document, assessment of the stability of the material and suggestions for further reading.

A-1.2 Assessment of Document Stability

Section A-2 Naming in the Service Architecture

Naming in the Service Architecture is not based on a general framework for TINA naming and as such cannot be regarded as stable material for use in implementations. However, some basic principles that are presented in the chapter are seen stable and can be used as guidelines for naming in the Service Architecture:

- The procedure to work with names in the information and computational view;
- The mapping alternatives between information view and computational view naming;
- The identified Naming Domains that need inter-domain naming schemes;
- The usage of X.500 naming scheme templates for Information view naming;
- The use of Administrative Domain Id to make Information Object instance and type names global;
- To allow several instance naming schemes for User Id.

Section A-3 Management

The material provided will be worked on further, by the core-team and by the service management work group. Parts of the material is likely to be placed in the main body in a later release.

Section A-4 Examples for Composition and Federation

This is examples of dynamics that was felt not to be stable enough to go into the main body of the document. It is likely that TINA core-team will continue work on this issues, in particular in relationship to RtR and 3Pty reference points.

The material is included to enhance readability of the main body, as they do not only include dynamics, but also illustrates the use of composition related information objects like CmpSR and ShSR.

Section A-5 Some possible refinements of service components

Section A-5.1 shows one way of handling service generic and specific behavior. Other ways exists al well, so this chapter must be seen as an example. As such it is quite stable. TINA core-team will not work any further on this issue.

Section A-5.2 User domain refinements. The relationship between TINA components and non-TINA components is unstable. No work has been done to define an API between the TINA and non-TINA part.

Section A-5.3 Additional service components. The TSA is unstable material.

Section A-6 TINA Common Services and Facilities

This chapter is unstable. It must be revised according to evolvement of composition and of the RtR and 3Pty reference points.

A-1.3 Issues requirering further work

This chapter describes some issues related to the content of this annex that still remain unsolved. Some of these issues are likely to impact the prescriptive part of the architecture, although it is not envisaged that they cause major changes; they are discussed in Section 9 of the Definition of Service Architecture part of the document. Some other issues are likely not have impact on the prescriptive architecture, and are discussed in this chapter.

The subscription model needs refinement work, both at the information and at the computational level.

TINA needs more work in the area of naming conventions. Several examples of naming trees are given in Section A-2, but none is enforced. Foremost, Terminal Id and Service Resource type and instance naming need further work.

A-2 Naming in the Service Architecture

This chapter describes the naming domains requiring a naming scheme that is common for several administrative domains. It also suggests naming conventions to be applied to these naming domains. Firstly, however, the main principles for naming in the TINA Service Architecture are presented.

A-2.1 Scope

Naming in the Service Architecture relates to the information and computational viewpoint of TINA. Naming in the technology and engineering viewpoints are of local concern only and will not be dealt with in this section.

Names are used to identify, locate and act on objects. The separation between the Resource and Service Architecture naming domains will not be total, i.e. certain names and naming systems will be common to both architectures.

- The Computational Object (CO) interface instance references will be common and rely on CORBA Interoperable Object References (IOR).
- The CO type naming systems may or may not differ between naming domains and architectures.
- Some Information Object type and instance naming systems will be the same for both architectures.

An example of the latter is a service session with a special adapter. The endpoints of this adapter should have the same naming scheme in the Service Architecture as in the Resource Architecture.

A-2.1.1 Information Viewpoint

The information viewpoint includes information object type and information object instance naming. The name of a standard for audio conference is an example of an information object type name, while the name of a particular run-time instantiation of the audio conference service is an example of an information object instance name.

The naming of information objects such as services, users and subscriptions are separated from CO interface names as they may have different life cycles. The CO interface IOR depends on both host and process, and of allocation of the object in the server (maybe inactive, so not a UNIX process). A process' normal or abnormal termination will cause the IOR to be useless and subsequent use of the CORBA IOR to cause errors. If the information object names are defined as attributes to a CO interface and published via a Trader, then an update of the IOR can be done either by the exporter explicitly or by the DPE implicitly (persistent object services). Importers will always request the service using the constraints formulated based on the information object names and in such way get hold of the most recent CO interface name (IOR). CORBA persistent object service can help to keep the persistent state of an object in case of abnormal termination and launch. CORBA migration object service can support the same thing in case of normal termination and launch.

A-2.1.2 Computational Viewpoint

In the computational viewpoint, the information object type and instance names, can be used directly as CO interface type and instance names, e.g. {audio_conference_SS} is bound to {audio_conference_GSS}, or occur as attributes in operations towards a CO interface. If many information object instances of the same type are expected, then it is preferred to hide the multitude of information objects behind one aggregated CO interface to achieve simplicity and lower ORB overhead with too many referenced CO interfaces. The top of the information object name structure

is then bound to the CO interface name.The information object name is uniquely defined as a concatenation of the CO interface and the information object name, where the CO interface name is unique within the trader domain and the information object name is unique at least within the CO interface domain.

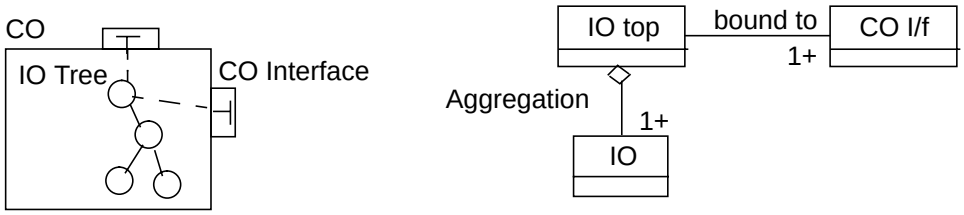


Figure A1-2-1. Information Object/Computational Object Binding Alternatives.

In most cases, CO interface instances will be identified by its CORBA interoperable object reference and no explicit naming is needed in the Service Architecture. However, in case of dynamic invocations, a CO interface instance name can be useful. An example, where this can apply, is in dynamic service compositioning.

A-2.2 Information Viewpoint

A-2.2.1 Naming Domains

A naming domain is a set of distinguishable entities that are named by the same naming authority(ies). Only naming domains that span more than one administrative domain in the TINA service architecture will be discussed in this section.

The following section identifies the information objects in the TINA Service Architecture that need to be named between administrative domains. The foundation is the administrative domain itself:

- **Administrative Domain**: An administrative domain states scope of ownership and management authority. An instance name of an administrative domain (stakeholder) is unique in a TINA system. Each administrative domain in a TINA system can be assigned more than one unique type and instance name. Administrative domains are identifiable and locatable by:
  - instance name;
  - optional: type name;
  - TINA services provided;
  - TINA service resources provided.

Subordinate to the administrative domain, a number of specialized naming domains with global significance can be identified:

- **User Id**: A user Id instance name identifies a user's location within one or more administrative domains and is e.g.,used to invite users to participate in one or more TINA services. Multiple users may be located in the same administrative domain. A user id instance name is unique within an administrative domain. Complemented with the administrative domain instance name, the user id instance name can be used to uniquely identify and

locate the user in a TINA system and allow international identification of users, user mobility and user authentication. A user id type name is unique within a TINA system and used for identifying user characteristics.

- **Terminal Id**: A terminal id instance name is unique within an administrative domain. Multiple terminals may be located in the same administrative domain. Complemented with the administrative domain instance name, the terminal id instance name can be used to uniquely identify and locate a terminal in a TINA system and allow international identification of terminals, terminal mobility and terminal authentication. A terminal id type name is unique within a TINA system and used for identifying terminal characteristics.
- **TINA Service**: An instance name of a TINA service is unique within an administrative domain. Multiple TINA services may be available from the same administrative domain. Complemented with the administrative domain instance name, the TINA service instance name can be used to uniquely identify a service instance in a TINA system.
  TINA services also need type names. More than one administrative domain (stakeholder) can provide a TINA service with the same type name, if they have the same syntax and semantics. In such case, a service specification has to be agreed among the stakeholders (or in some standardization body). In order to allow independent rapid service introduction, a service provided in a TINA system is initially likely to be stakeholder specific (and provided by one stakeholder only). In that case the administrative domain's name may be prefixed to the service type name. Part of a service can also be named within the same naming domain as the TINA service. In fact, many TINA services are aggregated from several **service features**. The same naming rules apply for service features as for services. Services and service features are identified and located by:
  - type;
  - compatible types[1];
  - optional: instance name.
- **TINA Service Resource**: An instance name of a service resource is unique within an administrative domain, i.e. the name identifies a particular service resource (e.g. video or web page). Multiple resources may be available in the same administrative domain. Complemented with the administrative domain instance name, the resource instance name can be used to uniquely identify a service in a TINA system. More than one administrative domain (stakeholder) can provide a service resource with the same type name, if they have the same syntax and semantics. In such case, a resource specification has to be agreed among the stakeholders (or in some standardization body). Resources are identified and located by:
  - type;
  - compatible types;
  - optional: instance name.
- **Flow Endpoints and Network Access Points**: (defined by the TINA Resource Architecture)

The following information objects does not necessarily have a global significance in a TINA system, but may be published by a stakeholder to other stakeholders.

1. A compatible type is one that can be used interchangeably with another type, i.e. supporting compatible interfaces. A specific example is where a subordinate object instance can replace a superior object instance.

- **Service Information**: This object contains a general description of the service, its properties, required software in the client equipment, etc. Information content is specified by the provider of the service. Example of information include:
  - Purpose of service;
  - Time(s) the session is active;
  - Information to receive the service (addresses, type of media and format);
  - Information about bandwidth and QoS to be used by conference;
  - Contact information for the person responsible for the session and/or address for further information about the session.
- **Resource Information**: This object contains a general description of the service resource. Information content is specified by the provider of the service resource. Example of information include:
  - Time to live constraints;
  - Format;
  - Version;
  - Costs;
  - Ownership;
  - Location;
  - Security mechanism;
  - Binding principle.

Other information objects, as e.g. the information objects used in the service session graph model, are given a locally defined instance names by the administrative domain acting as a provider in the user/provider relationship. No naming convention is prescribed for these information object instances, but often a type (class) based instance naming suffices, i.e. the instance name is relative to the class and the session. The rest of the intelligence is in the service session graph structure (configuration) itself, e.g. the parent - child relations are stored separately (see also A1-2.2.2.3.4.).

A-2.2.2 Naming Conventions

The information viewpoint uses quasi-GDMO/GRM templates for defining naming systems. The reader is referred to [Naming Framework document] for a detailed description of the templates.

Three names are needed for each object: object identifier, object type label and object instance name. The template id is used to register the type description in a type repository. The other two are used within programs to denote the type and instance respectively.

A1-2.2.2.1 Object Identifier Naming System

For each type description, there is basically three possibilities to give it a unique template identifier. The template identifier (an ASN.1 object identifier) can be derived from:

- an existing standard;
- a TINA defined standard; or

- an administrative domain specific specification.

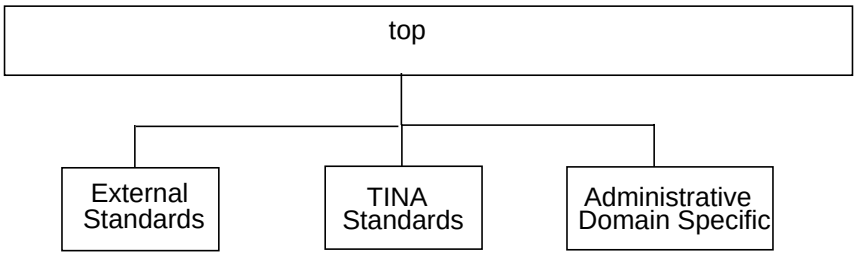


Figure A1-2-2. Template Naming Hierarchy.

External standardization bodies will be used to identify grandfathering (embedding) of existing standards. It is expected that a new object type shall be possible to register automatically in a TINA type repository in order to allow an administrative domain to publish new object types without standardization.

A-2.2.2.2 Type Naming Systems

Two main naming systems are applied in the TINA architecture:

- Name binding making inheritance relationship between object types visible;
- Name binding making an aggregation[2] relationship between object types visible.

Subtyping is often used in order to refine the semantics and the syntax of an object type, e.g. to define several instance naming conventions that can be used for the object type.

A1-2.2.2.2.1. Administrative Domain

A type naming network for administrative domain may be established in a similar way as the top level domain names in Internet, e.g. ‘com’ and ‘edu’ in the USA, specializing on the social functions of the administrative domain.

2. A special notation has to be introduced in the X.500 templates in order to cope with type naming based on aggregation relationships between types.

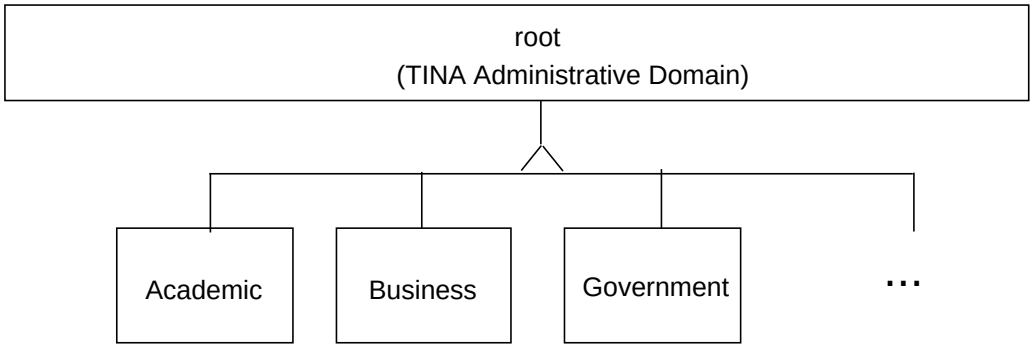


Figure A1-2-3. Example of Inheritance Relationships between Administrative Domain Types in the Service Architecture from which a Type Naming Network can be Derived.

A1-2.2.2.2.2. User Id

The user information object is subtyped into subscriber and end-user in TINA.

The user may be further subtyped depending on service context based instance naming scheme, i.e.:

- user name
- e-mail address;
- phone number;
- facsimile number;
- paging number;
- URL to the user's home page.

A1-2.2.2.2.3. Terminal Id

No definition of subtypes for terminal is specified yet.

A1-2.2.2.2.4. TINA Service

Type naming of a self-contained service or service feature is based on type inheritance (see the service classification in Section A-7.1) and/or grandfathering from existing service standards. The following service types are initially grandfathered under the TINA umbrella:

- ITU-T ISDN bearer capability, high-layer compatibility and supplementary services [33].
- ITU-T B-ISDN broadband bearer capability, broadband high-layer information and broadband supplementary services [34] [35].
- ITU-T IN services and service features [31].
- IETF IANA well-known service types [41].

It should be noted that most of these service naming conventions relate to a specific transport technology.

For static service compositioning, a type containment based naming scheme may be added in order to denote which compositions that the service feature is designed to cope with (service interaction consistency concerning both data and logic).

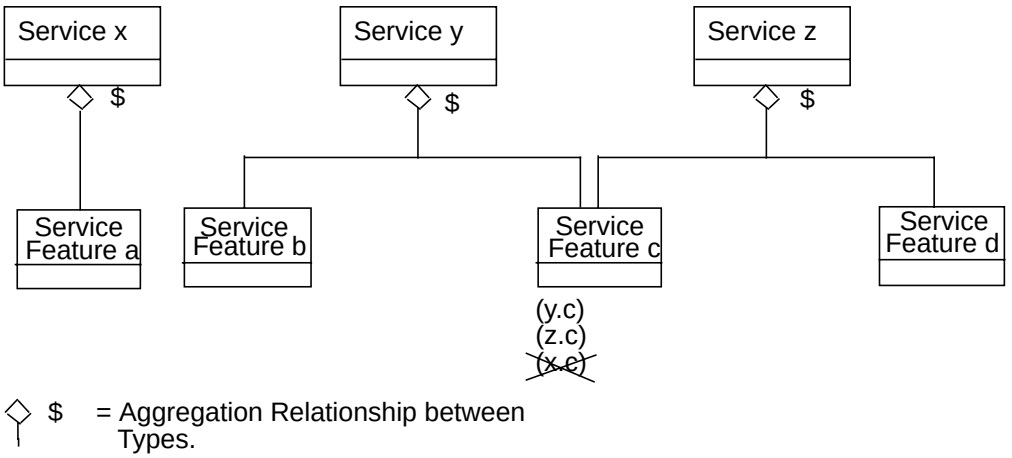


Figure A1-2-4. Service Type Naming Based on Type Containment Relationships.

In the example above, the type name (y.c) tells the composer that the service feature c is designed to be combined with service feature b to perform service y. As service feature c has not been designed to interact properly with service feature a, it has not been given a type name (x.c). A name comparison rule for the different names of the same entity need to be defined.

A1-2.2.2.2.5. TINA Service Resource

The resources in the Service Architecture can be subtyped based on instance naming conventions and context as shown in the example below.

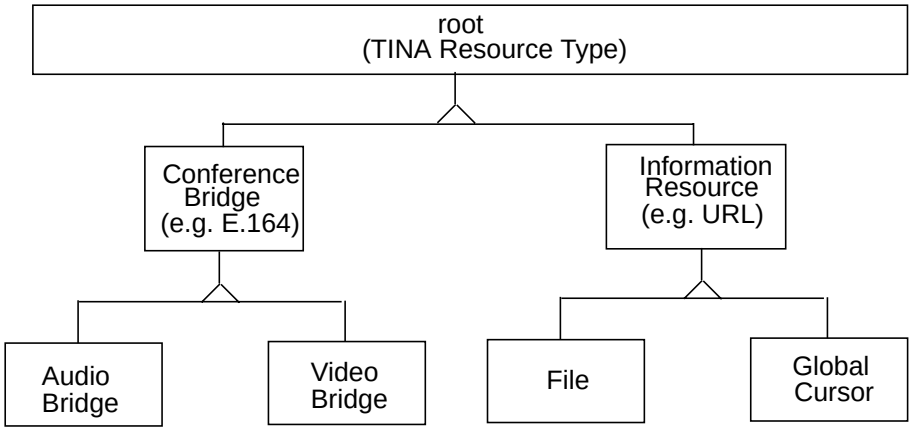


Figure A1-2-5. Example of Inheritance Relationships between Resource Types in the Service Architecture from which a Type Naming Network can be Derived.

A-2.2.2.3 Instance Naming Systems

The main requirement on instance names is that they are unique within a defined scope. A locally unique instance name can then become globally unique by using an administrative domain identifier in the information view.

An object instance may have several names or a name consisting of a concatenation of attributes. In order to avoid naming collisions, aliases rather than multiple inheritance shall be used to model object instances when multiple names exist.

Two main naming conventions shall be applied to information object instances in the Service Architecture:

- Name binding making containment relationships between object instances visible;
- Name binding making type relationships visible for the object instance.

The name binding may reside either in:

- the local computational object where the information object resides;
- a DPE naming server;
- a DPE trader.

The distinguished names will only be used when crossing administrative domains. Relative distinguished names can be used within local domains.

As an alternative to static name binding, external relationship objects can also be used to maintain the relationship between information object instances in a distributed system. The naming systems for the related instances can then be independent, but can also be enforced to have the same fixed length for names, etc. The performance of a hard coded name binding shall preferably be used for performance critical information object instances.

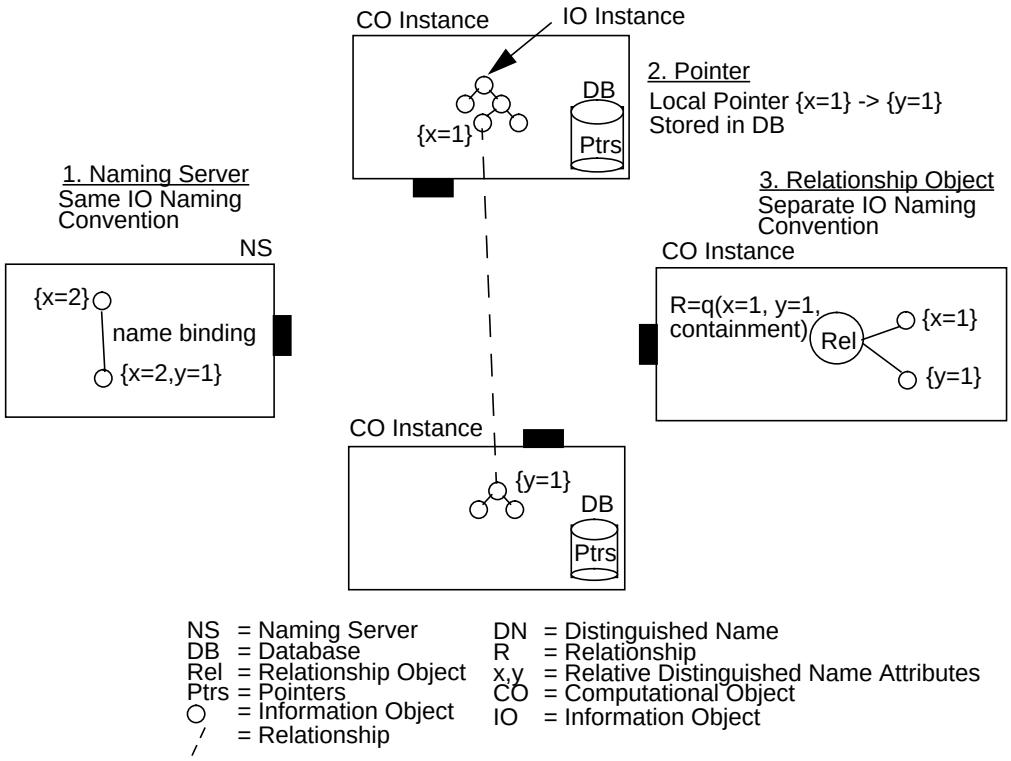


Figure A1-2-6. Three Alternatives for Maintaining an Information Object Instance Relationship Hierarchy.

A1-2.2.2.3.1. Administrative Domain

There are three essential characteristics of administrative domains that need to be taken into account when defining a naming convention:

- object oriented;
- high dependency on data;
- strong relation to location.

Many large stakeholders want to have a simple, global name and, therefore, dislike the idea of using country code (ISO 3166) as a designator in the instance name. The local laws on trademark disputes, contentious names, etc. can, however, be applied more easily if a country code designator is used. A scheme supporting the latter proposal is, therefore, preferred.

The second question is if also social functions should be used for administrative domain instance naming in the same way as top level domain names are used today in the Internet, i.e. edu and com in the USA. It is questionable if this split in instance domain name spaces is good. Many large stakeholders can have several social functions but still want to remain with its administrative domain instance name within one name space. In a TINA system it is seen as preferable to manage the social functions as an optional search attribute separate from the instance name.

The following object classes from ITU-T X.521 [38] can be reused to define an administrative domain name[3]:

- country;
- organization;
- organizational unit.

The name binding structure rule is preferably based on the containment relationships between the objects instances.

A1-2.2.2.3.2. User Id

The user id is based on a further containment relationship from the administrative domain instance name. The total directory information tree is typically: root, countries, organizations, organizational units/users.

The user id instance naming scheme shall have the following characteristics:

- not tied to transmission path;
- registered with an administrative domain;
- globally accessible;
- portable between administrative domains within a country.

The scheme satisfying the above would be a UPT naming scheme. However, awaiting these standards, and in order to allow interworking with existing networks, it is proposed that a user be identifiable with several user ids. The information to be maintained about a user by the administrative domain in a TINA naming service should at least include:

- The user's name;
- The user's service context dependent names:
  - e-mail address;
  - phone number;
  - facsimile number;
  - paging number;
  - URL to the user's home page.
- Some indication of the User's relationship with the administrative domain;
- The user's postal address.

The definitions shall be according to ITU-T X.501 [36], X.520 [37] and X.521 [38] standards.

When the user id is ambiguously specified, the administrative domain should provide user-friendly means of discriminating between entities. For example, it might require first name and middle initial in addition to last name for the user's name (see X.501 Annex H). The commonName attribute which hold the common name of an organization or user should contain all known aliases for the organization or user. Every entry has a relative distinguished name from one global name space.

A1-2.2.2.3.3. Terminal Id

The terminal id naming scheme can either be based on an existing standard, such as GSM equipment id, or be a new TINA standard, e.g. a generalized ORB id.

3. The ASN.1 datatypes used in X.500 needs to be aligned with the IIOP Common Data Representation Transfer Syntax in CORBA 2.0.

A1-2.2.2.3.4. TINA Service

There are three essential characteristics of the naming convention for services and service features.

- It is functionally related (including capability set/snapshot notation).
- It has low dependency of data.
- It is independent of location.

Service and service feature Instances are, therefore, mainly named based on a service type naming scheme (including version handling), i.e. type_name/type_name/instance_name. The following shows an example for a freephone service:

IN/cs1/freephoneService/admDomain/unique service instance name defined within the AdmDomain

A1-2.2.2.3.5. TINA Service Resource

Only the URL schema is discussed below. Other schema may be defined or adapted later, e.g. a protocol independent Universal Resource Name (URN) scheme or an instance naming scheme for other service resource types.

The X.500 URL attribute type specifies the URL associated with an information object instance. An attribute value for URL is a string that complies with the Common Internet Scheme Syntax for Universal Resource Locators (URLs) [42] [43]. The URL syntax is:

- <scheme>//<user>:<password>@<host>:port>/<url-path>
- <url-path> points to file

The URL attribute is protocol dependent via the scheme syntax. Already defined URL conventions are:

- ftp, http, gopher, mailto, news, nntp, telnet, wais, file, prospero, whois++, data and z39.50.

An additional scheme for CORBA IIOP may be adopted later.

A-2.3 Computational Viewpoint

A-2.3.1 Naming Domains

Computational object interfaces, objects and object groups are identified and located between domains by:

- type name;
- compatible types;
- optional: instance name.

Some, but not all need to be exported/imported between administrative domains. In the service session, UAP, USM, SSM, and SSO are service components[4] that shall be visible and locatable between domains. In the access session, provider and user agents are others. In particular, these session objects shall be able to locate each other on session instantiation, resumption and recovery.

4. A service component is an object or a group of objects used at service compositioning.

The related information object type and instance names are not mapped directly to the above computational object type and instance names, but will rather be used as attributes in operations. The reason being that the user id will identify the user at the user agent as well as the party at the service session manager. Similarly, the service name will identify the service at the user agent, the service session manager as well as the accounting management object.

A-2.3.2 Naming Conventions

A-2.3.2.1 Type Naming Systems

Computational object interface types can be uniquely identified by an inheritance classification scheme agreed within the TINA community. Alternatively (or as well), TINA allows heterogeneity for type naming, by using an OMG Trading Object Service [45]. Types are then defined using the OMG object trading service format which associates a trading service type with an interface type and a list of properties. Only the trading domain is required to be uniquely identified between administrative domains.

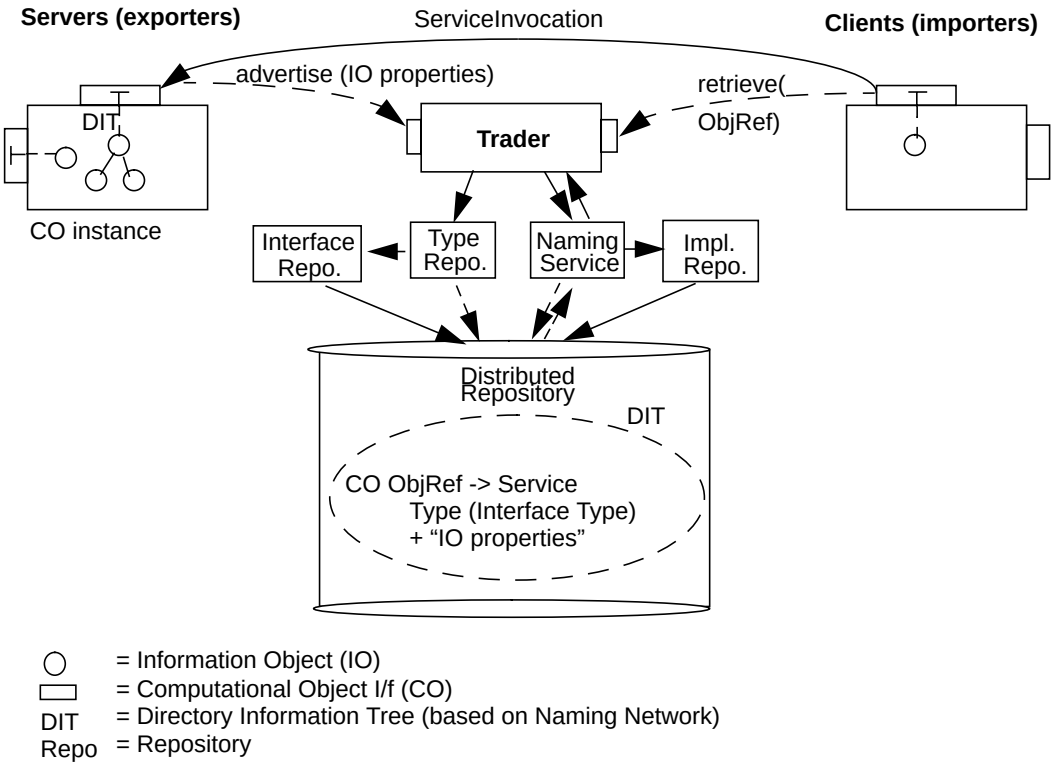


Figure 2-8. Using Trader for Information Object Name Resolution.

Trading service types can be related in a hierarchy that reflects computational object interface type inheritance and/or the property type inheritance/aggregation. The information object type naming can be used as property value constraints in trader searches. If made mandatory, they can become the actual information exported/imported.

A-2.3.2.2 Instance Naming Systems

Computational object interface instances will be supplied with an identity by the DPE, a CORBA interoperable object reference [46]. Some computational object interfaces may be given instance names by the Service Architecture application in order to facilitate searches. The CORBA naming service allows the Service Architecture application to add instance names to pseudo-objects which in turn are mapped to computational object interface instances. A computational object instance name is then only allowed to be bound to one entity, while one entity may have several instance names.

Several naming conventions for computational object interface instances exist:

- Type based: type/type/objectInterfaceInstance;
- Aggregation based: namingDomain/groupInstance-1/subGroupInstance-2/objectInstance/objectInterfaceInstance

The latter variant is preferred in the Service Architecture to facilitate management of objects and object groups.

The DPE may be structured in DPE naming domains and different naming conventions may apply for each naming service. In this case, a unique naming domain id must be specified and the computational object interface instance names need to be uniquely identified within that domain.

Implicit or explicit exporting of names may apply between naming services (see figure below). Explicit replication of data between naming services is recommended to improve the performance. In order not to violate privacy laws, it is recommended to restrict replication to country and organizational entries and knowledge references (which tells where to go for which part of the naming service). Local replication between two databases within the same organization is highly recommended.

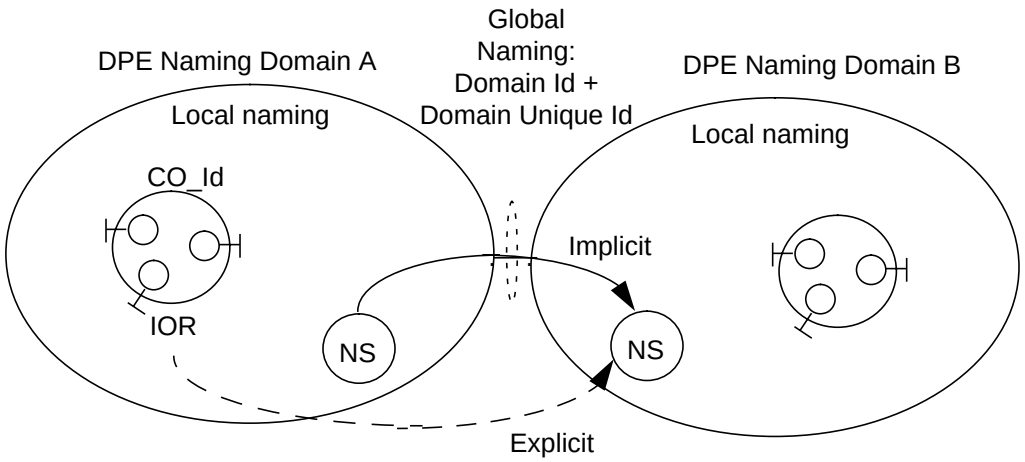


Figure A1-2-9. Federation between DPE Naming Domains for Computational Object Instance Names.

The mapping between a information object instance name and the computational object instance name is not always one-to-one. An information object instance name may be mapped to an information object_id (which could correspond to a thread_id in the engineering view). In this way, a flat naming structure could be maintained for information object_ids, still keeping the hierarchical,

human-friendly information object names towards the applications. The benefit is a freedom in allocation of information objects in the distributed system. The cost for this allocation freedom is performance as a separate directory server needs to maintain the mapping between information object name and information object_id.

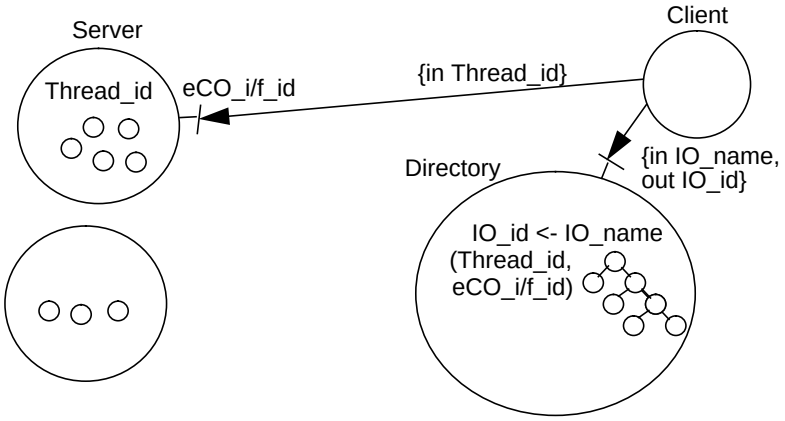


Figure A1-2-10. Mapping between Hierarchical Information Object Names and Flat Information Object_ids (CO_id or thread_id) via Directory.

Multiple information object instance names are possible to represent within the same computational object interface. An information object instance name can either be expressed as an attribute or as an argument in an operation. If an information object `<rdn-attribute>` is directly mapped to a `CO_interface_type attribute`, the component specifier should take care of any overloading of `<rdn-attribute>` made in the information model.

```
//ODL
interface CO_interface_type {
    readonly attribute Name n;

    void operation_type (
        in Name n
    )
};
```

The `operation_type` represents an operation type of the information object assigned to the computational object interface.

The information object instance names can also be used as constraints in trader searches, but are difficult to maintain as they are changing very often. Therefore, some alternatives are to:

- Let the computational object instance resolve the information object instance name via a trader Service Offer Evaluator (SOE) interface;
- Let an external computational object act as a dynamic property evaluator and control the information object tree and interact with information object instances in computational object instances;
- Let the destination name be well-known to the client and use it as property value constraints in the request to the trader.

The two first alternatives have implications on the performance of the trader and should be applied with caution.

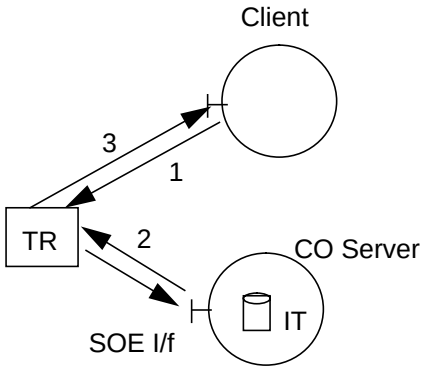


Figure A1-2-11. Service Offer Evaluator Interface (SOE) for resolving information object instance names. The Service Offer Evaluator Interface is used by the trader to resolve unknown properties value constraints.

A-3 Management

This chapter provides additional information which helps understanding of Section 5. The material is considered useful, and more or less stable. The material consists of either supplementary explanations and examples to material in the main body, or material which we have been unable to tie into the rest of the service architecture more concretely like the service group. For this reason, there is no clear structure in this document; this section is rather a compilation of useful material related to the service management.

In the first section (Section A-3.1), we provide usage analysis of service management; it is hoped that this section explain how the concepts exposed in the main part are being used.

From Section A-3.2 on, we explain other useful concepts in TINA service management framework, such as Pick and Mix, Service Groups etc. The information model of management context is explained in Section A-3.5. Service customization issues are discussed in Section A-3.4. The negotiation phase of management context, i. e. the general context configuration management, is discussed in Section A-3.6. Materials from life cycle management, both service life cycle and user life cycle management are explained in Section A-3.7 and Section A-3.8.

A-3.1 Usage Analysis of Service Management

In the main body of service architecture, we took a static, mostly information viewpoint to describe the management of the TINA service architecture, i. e. principles and definitions of main concepts are exposed along the management axes. In the following sections, we focus on the dynamic aspects of the service management. Our intention is to illustrate the MgmtCtxt and its usage, in particular how it is configured, bound to a service session, and then interpreted to be a part of the session management (SSM, USM) using an example. As it is evident from the title of this section, we call this timewise analysis of the service management framework as a process analysis, since it constitutes several recognizable semi-independent steps (processes), to proceed from the configuration to the operation.

Figure A1-3-1 illustrates the service management processes. The first two processes are preceding to the service session, and the last three processes are performed as separate phases within a service transaction.

1. *Context Configuration Management:* management contexts are configured through GCCM, a service component that deals with configuration and negotiation of management contexts.
2. *Terms of Management:* ToM is established, which is bound to the service session subsequently created by the user, or joined by the user, if the service session already exists.[1]
3. *Set-Up:* set-up phase of the service transaction. Associated management contexts are first interpreted, followed by service components (mainly USM and SSM) set-up and resource components (e. g. Event Management and Log) set-up. If the service session consists federated or composed parts, they have to be set-up as well.
4. *Execution:* the service transaction, which is also a part of the service session, is executed.
5. Wrap-UP: the performance (QoS, QoP) measured during the execution phase is summarized. Recovery actions may be taken if the measured performance have not achieved the guaranteed quality. After the service transaction is concluded, the resources are released and then reclaimed by the management system.

1. This is typical when the service session represents a multi-party conference.

Although not shown in the figure, there is a business relations between the business administrative domains, which can be seen as pre-contexts of whole the service management activities. In the following of this section, we describe these processes..

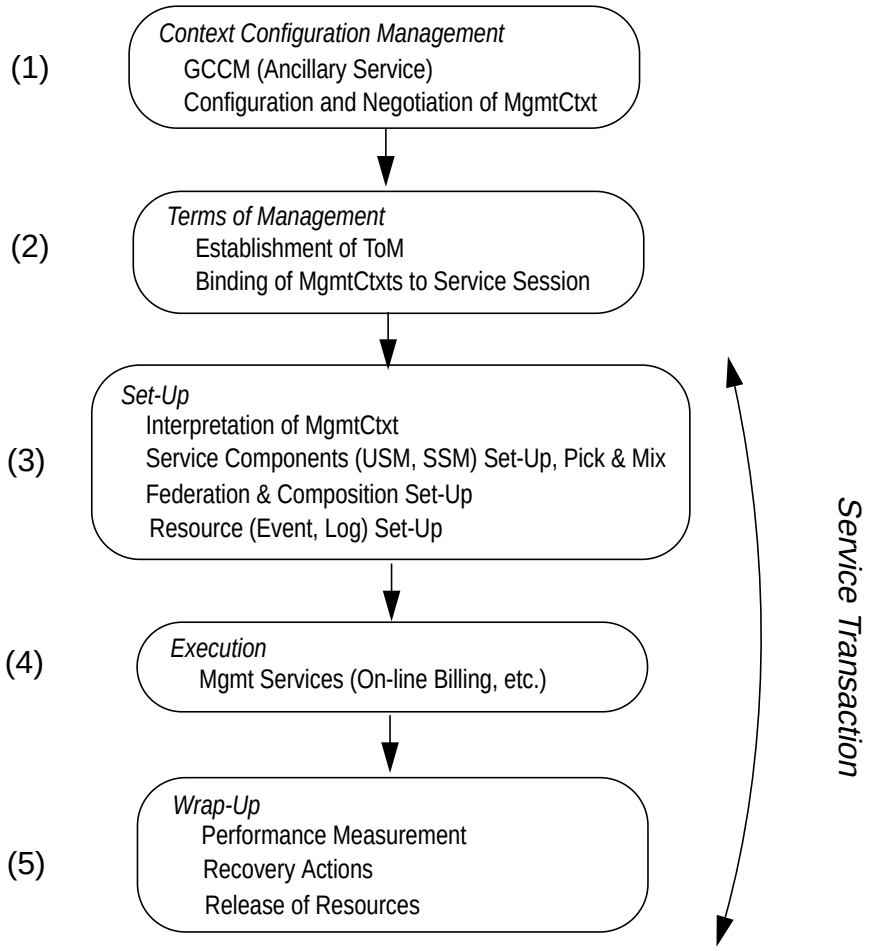


Figure A1-3-1. Service Management Processes

A-3.1.1 Business Administrative Domain Relationship

Business Administrative Domains (see Section 3) and their business relationships define pre-contexts of service management, as well as many other issues in the TINA service architecture. In particular, they define the user-provider relationship between the administrative entities, upon which the service transaction and other service management activities are performed.

Figure A1-3-2 explains the service transaction concept applied to a simple three-party accounting transaction. In this example, three parties, User, Network Provider (NP), and Service Provider(SP) are involved, and the billing flow between the three parties are shown. Although the total amount of charge to the user is the same amount of A+B, the billing flow can differ depending on the business relationship between the three parties. In case (1), the user starts two separate transactions with NP

and SP. In case (2), NP acts as an integrator, thus NP receives a bill from SP, and sends the bill to the user adding its own bill. In case (3), SP acts as an integrator. In cases (2) and (3), two transactions are nested, and the user is involved with the third-party only through a nested transaction.

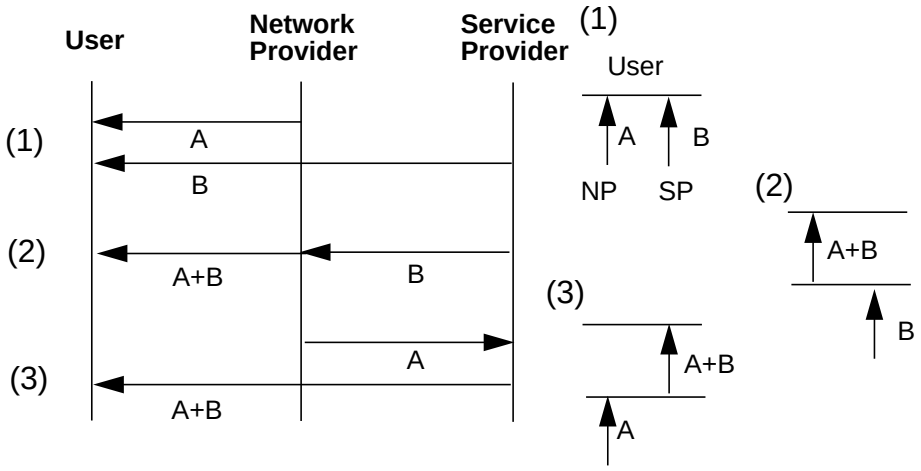


Figure A1-3-2. A Simple Three-Party Accounting Transaction

It can observe from this example that all three cases can be handled by the service transaction concept, and only difference is that the nesting relationships between the transactions. Consistent service management, for example on-line charging, is also possible, by passing a MgmtCtxt to the nested transaction. In case (1), it is apparent that the user can set his/her accounting requirements in a MgmtCtxt and by giving it separately to two the two providers. In cases (2) and (3), the user's MgmtCtxt is passed from the integrator to the third-party provider, thus making it possible for the user's accounting requirements to reach the third-party provider, allowing a consistent service management throughout the multiple business domains.

By including wrap-up phase as its integrated part, service transaction automates guaranteed QoS (QoP[2], accountability, etc., depending on which of FCAPS context is the subject) concept by coordinating FCAPS management with the billing process. In simple words, the user is not charged if he/she did not receive the service of guaranteed quality.

As we've seen in this example, all three cases are roughly equivalent, as far as the billing goes to the end user, i. e. the amount of the bill is the same, and the same billing scheme (service transaction and the management context) is used. The difference, however, arises from the business relationship between the three parties, from which the nesting relationship of the two service transactions is derived.

A-3.1.2 Context Configuration Management

The concept of context is widely used in TINA service architecture in various different forms. Examples include usage context, accounting management context (AccMgmtCtxt), and security management context (SecMgmtCtxt). The major purpose of these contexts is to associate additional information, represented by a context, with a service instance, so that particular service objectives

2. Quality of Protection.

are achieved. For example, the usage context is used to realize terminal mobility [23]. The management contexts (AccMgmtCtxt, SecMgmtCtxt etc.) are used to achieve service management goals such as guaranteed accountability and quality of protection.

In any case, despite the difference in their usage and purposes, all these contexts are strikingly similar in their structure and the way they are configured in the service architecture. They need to be agreed between the user and the provider, therefore negotiation is often necessary. General Context is a generalized class of context in the service architecture. A service component called GCCM (General Context Configuration Manager) is conceived to handle the configuration and negotiation phase of context set-up. Since management context is a sub-class of the general context, the configuration and negotiation of management contexts is handled by GCCM as well.

A-3.1.3 Terms of Management

Since ToM consists of implicit part (parts of user profile and subscription contract) and explicit part (management context instances), the establishment of ToM means both. For the implicit part, there needs no actions, since default binding of the implicit part is (mostly) under the discretion of the provider. On the other hand, binding the management contexts can be seen as explicit agreements between the user and the provider, as such the user can specify the management contexts to be bound to the subsequent service transaction.

A-3.1.4 Set-Up

The set-up phase of the service transaction is perhaps the most involved as well as the most contrived of the five service management processes, since it has to configure or re-configure service components as well as resource objects to implement the management functions dictated by the management context. As such, it can not be explained without dealing the internal structures of USM, SSM, and the service session graph (SSG), which represents the relationship between the service session and the session participant in the form of a graph (as explained in Section 6.4).

- *Interpretation of Management Context:* management contexts are interpreted by the service session, and is subsequently implemented within the session via the service components set-up.
- *Service Components Set-Up:* when management contexts are bound to the session, internal configuration of USM and SSM may need to be re-configured to accommodate management needs. For example, accounting events (or billing info.) need to be passed to the user when on-line billing is requested. USM therefore has to be reconfigured such that it opens an event channel to the user, over which the accounting events are sent.
- *Pick and Mix:* although its applicability may be limited when the service components are just re-configured, not created from scratch, the concept can still be used when some internal configuration of SSM or USM has to be changed, i. e. resource at the service level is dynamically allocated or re-allocated. For more information on pick and mix, please see Section A-3.2.
- *Federation and Composition Set-Up:* federation and composition relation must be set-up as well, for two (or more) service sessions to work together to achieve some common management objectives, e. g. QoS of the whole service sessions.
- *Resource Set-Up:* much of this phase is related to network management, not to the service architecture. In particular, the configuration management of network resource is the essence of the resource set-up. The reader is referred to the corresponding section in NRA [8].

Among the above set-up issues, we focus on the two conspicuous ones, namely the service components set-up and the federation and composition set-up. Since the interpretation of management context is rather a trivial issue once the exact syntax of a management context is determined, we have to wait till the management requirements in the respective FCAPS field are fully identified.

A-3.1.4.1 Service Components Set-UP

Once a given management context is interpreted, it is translated into internal configuration of service components. The way the service components are re-configured, however, depends on the management context, and it can not be explained without using an example. In general, however, they act on each sub-components of USM and SSM in a specific way. Presumably, it should be possible to divide the (interpreted) context into two parts, a private context and a common context. A private context is a part of the context which only affects the management functions of the individual user, not the whole of the session. In the following example, we assumed that the on-line billing is part of the private context, since any participant may opt to have on-line billing without interfering other participants' management requirements.[3] This separation is a matter of convenience, since it will allow easier maintenance of private contexts; if the context is private, it is ok for a USM to create or delete the context without consulting its associated SSM.

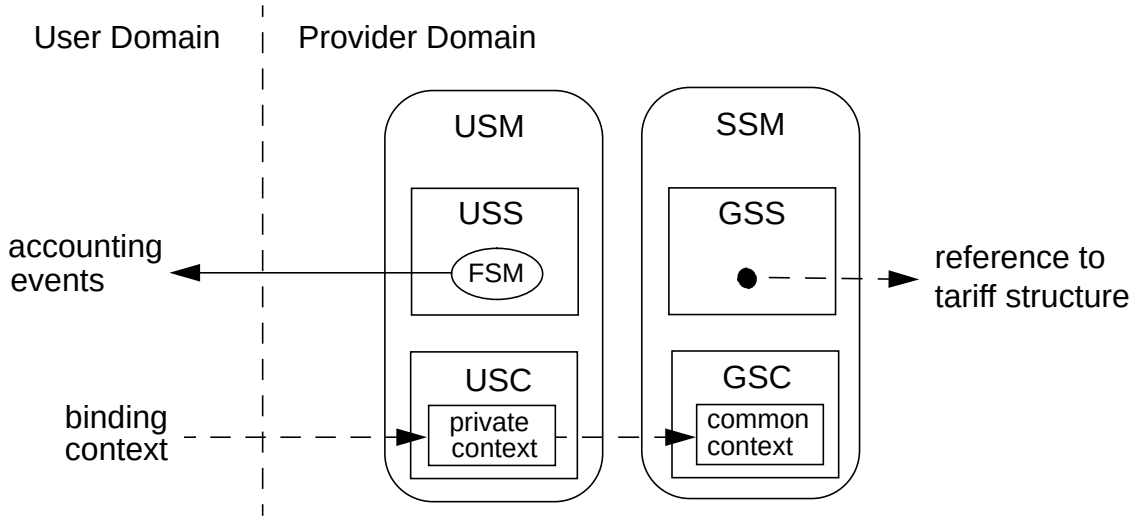


Figure A1-3-3. Private and Common Context in An Accounting Example

Figure A1-3-3 illustrates the roles of private and common context in an accounting example. For the sake of simplicity, interfaces of the service components (USS, USC etc.) are not shown. The binding context of the provider (and also the user) is the logical sum of the two contexts, the private context and the common context. Therefore it is absolutely necessary that the negotiation process within the access session concludes in such a way that the binding context is consistent and executable by the USM and the SSM. Service specific part of the private context is executed by USS. In this example, an on-line charging is required by the context. When the USM is instantiated (as it is explained in the

3. It is however possible, though it may be an exceptional case, that some property (or a session relationship) of the service session virtually prohibits on-line billing to its participants. In this case, the on-line billing can not be considered as a private context.

next section on the Pick-and-Mix principle), Service Factory dynamically attaches a finite state machine (FSM, perhaps implicit within the application itself), which generates accounting events. These accounting events are immediately sent back to the user using an event channel, which is also defined in the (private) accounting management context.

GSS also holds service specific part of the common context. For example, it seems appropriate that the reference to the tariff structure of this service session is contained in GSS, such that each session participant can access the price structure being applied to the session, through GSS.

3.1.4.2 Internal Representation of Common Context

The relationship between management contexts and Session Graph (SG) requires special attention. In the original concept of SG, it was once considered that the information objects describing the service management (the current management contexts) are part of SG. In this version of the Service Architecture document, we distinguish management contexts from SG, mainly for the following two reasons:

- Though they both are related, and they both eventually are executed by SSM, their roles in TINA services are different. Since SG is given only information model at this point, this separation allows us to study service management issues separately from SG issues. Besides, this separation does not harm any integrity of the TINA service.
- Since MgmtCtxts are used to create (or customize) USM and SSM using the *Pick-and-Mix* approach explained in the next section, MgmtCtxts have to exist before SG.

SG however is an ideal place to store properties common to all the session participants. In general, a MgmtCtxt can be divided into two parts:

- Properties common to all the session participants
- Properties applicable to each individual participant (the user)

If we take an example from accounting management, a shared-billing can be considered a common property (or context), whereas an on-line billing can be considered a private context. In a multi-party service session, it is fairly easy to change a private context, since only the agreement between the user and the provider is necessary. On the other hand, it requires possibly negotiations and an involved agreement making process to change a common context.

A-3.2 Requirements-Driven Instantiation of Service Components (Pick-and-Mix)

TINA in a sense is a constant challenge for changing the telecom industry from the resource-oriented to the service-oriented, enabling itself to respond more quickly to dynamic, market-driven world. The same approach is consistently applied to the service management, such that the service management be more service-oriented. To fulfil this ideal, TINA service management requires greater amount of flexibility and dynamics than conventional approach. One of the key steps, which is the second step of the service management mapping process, is the requirements-driven instantiation of service components, which we call **Pick-and-Mix** approach in short.

In this Pick-and-Mix approach, a set of objects supporting the service management are grouped according to their FCAPS functionality. These objects are dynamically attached to service components when they are instantiated at service factory on demand, or in a requirements-driven (or context-driven) manner. This process also adds additional management or notification interfaces to the instantiated service components, as they often turn out to be necessary.

Besides the intended flexibility, this pick and mix approach provides additional merits:

- Management components are reusable, and they can be separately and independently maintained from service applications. For example, a TINA service application would be able to receive better security or configuration support from the future update of management functional components.
- Since management functional components are attached dynamically on demand, this approach may help minimizing the service management overhead of TINA service application.

Figure A1-3-4 illustrates the basic flow of the pick and mix approach:

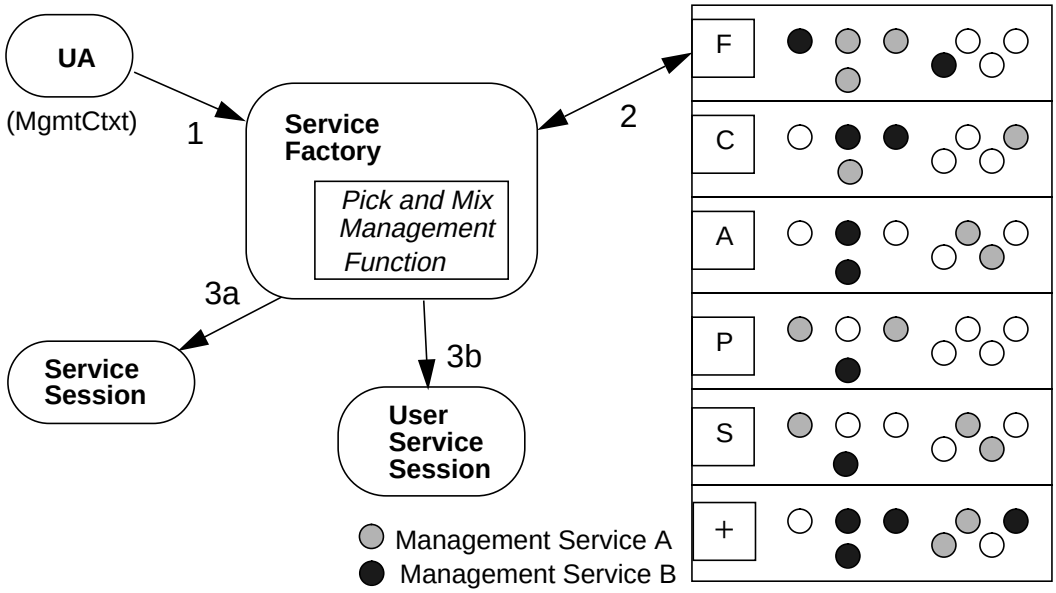


Figure A1-3-4. Requirements-driven Service Components Instantiation.

1. Service management requirements in the form of management context are passed to the Service Factory (1), where the management context is subsequently interpreted and necessary management functional components are selected.
2. (*Pick*) The Service Factory instantiates or provides references to the management functional components (2).
3. *(Mix)* The Service Factory then instantiates a Service Session Manager (3a) and a User Session Manager (3b). Management functional components are either instantiated within SSM and USM, or only pointers are passed to the respective service components.

The Pick-and-Mix approach is not without its limitation. Since the service and management interfaces will utilize the same data and have interacting logic within the service component, both the service components and the service management functions must know them at compile time of the objects in the service component. As such, management functional components to be used by the Pick-and-Mix must be carefully designed, such that the picked components implement the intended service logic without leaving unresolved reference to the data structures which the service logic may utilize. The technical challenge involved here is similar to that of a highly sophisticated linker.

It is however conceivable that the advancement of applet technology may allow us more dynamic resolution of the issue. For example, if the user requests that the billing be on-line in the middle of the service session, the events and their data structure the user is to receive from the provider must be understood by the user. To be more precise, they must have been understood by UAP when they are compiled. By applying the applet concept, part of the UAP can also be dynamically loaded as necessity arises, such that accounting event type and its involved service logic can be loaded into the user's side. The details of the implementation issues of the PIck-and-Mix approach is, however, still under study.

A-3.3 Service Group

To manage the complexity of the life cycle management, it is certainly necessary to provide a construct to organize service supporting entities such as service types and service instances. The construct needs to be a part of the life cycle management of the service architecture, such that any update such as creation of new service types and deletion of existing types be quickly reflected onto the configuration management supporting mechanism within the life cycle management.

In this section, we introduce **Service Group (SGP)** concept. Since deployment and withdrawal of a service implies a group of objects constituting the service need to be deployed or withdrawn, it is natural to bundle those objects as a service group [20].

The motivation for this requirement is best illustrated using a database example. Consider a database which serves a number of applications, see Figure A1-3-5 scenario. Within the database there is data which is used by Application A. A subset of this data is also used by Application B, (i.e. data at the intersection of the two applications in Figure A1-3-5). A possible representation of this scenario is shown in Figure A1-3-5.

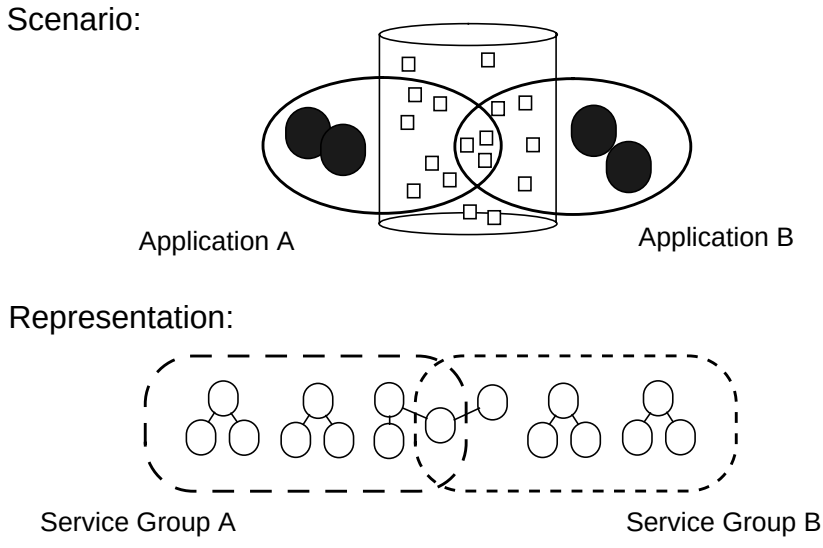


Figure A1-3-5. Two Service Groups Sharing an Object Group.

The two applications are each modelled as a service group. Each has an object group which models the database. The shared data is represented by a contained object group shown at the intersection of the two service groups.

Should Application A be withdrawn, then it is necessary to remove the data which is used Application A whilst leaving the data which is in the intersection of Application A and Application B. Thus when withdrawing service group A, the object group in the intersection of the two service groups must remain.

Figure A1-3-6 illustrates some of the particular properties of a service group. The figure shows five service groups A, B, C, D and E.

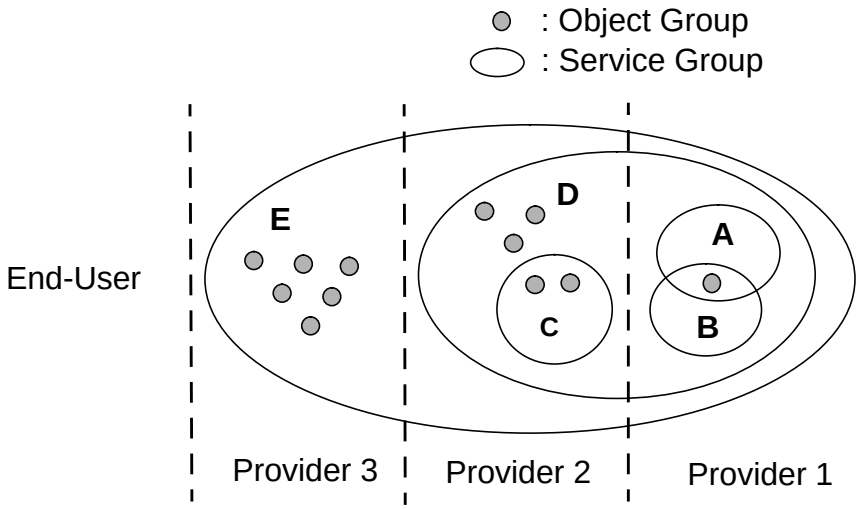


Figure A1-3-6. Service Groups in a Multiple Provider Environment.

The service, which service group E is the implementation of, is built (service composition) from service groups and object groups that extend across the domain of three providers. The figure illustrates diagrammatically:

- **Containment**: Service groups A and B share an object group. Service group E contains service groups D, which in turn contains service groups A, B and C.
- **Scope**: Service groups E and D spans multiple administrative domains. A service group may span one or more DPEs. The client of a service group does not see the contained service groups; Provider 3 only sees service group D when constructing E.
- **Dependencies**: Service group E is dependent on the prior installation and activation of service group D, which in turn is dependent on service groups A, B and C.

The “service group template” is the computational representation of service encompassing the binaries of the contained object/service group templates plus the additional configuration information. Upon receipt of a service request, a new service group is instantiated from the service group template. Service group is similar to the object grouping concept at DPE level, particularly both consist a similar containment hierarchy concept. They need to be distinguished, however, mainly because of the other two features above, namely a SGP may have a scope across multiple DPEs, and a SGP may contain a template for **deployment dependency** between object groups within the SGP. The last feature is particularly important, since it often occurs at the resource level that a certain object group need to be installed preceding the others, in order not to generate false alarms. The same deployment dependency can be usually re-used in the withdrawal phase.

It should be noted that the semantics of services and service sessions are modelled in the information viewpoint of TINA. The mapping towards computational interfaces, objects, etc. are done and can result in that a service only occupies a part of a computational object in order to lower CO overhead or to share common resources. An example is that many services will share the same CO resource manager of a bridge for their individual access points to the bridge. This means that a service mapped to the computational view may not control a whole object or even a whole interface.

The TINA DPE architecture is defining a general concept for DPE domains as a collection of CO interfaces, including configuration management domains. It is proposed that the ideas of service groups are generalized and added to the DPE architecture[4], particularly to the object grouping concept [18], which still awaits further study.

A-3.3.1 Service Group Life Cycle Management

It is apparent that an SGP may have its own life cycle separate from the service the SGP is associated with. It comes from the nature of the grouping concept, i.e. a SGP may be reusable for another service even when the original service the SGP was deployed for, may no longer in use.

It is probably natural to assume that an SGP is also a unit of service composition. So far, the relationship between the SGP and the service composition has not fully investigated, but interested readers are referred to [20].

Although the life cycle of an SGP can be separate from that of the service, by definition they follow the same life cycle model; the same life cycle states in the deployment/withdrawal phase. For this reason, we assume that life cycle management of SGPs are done by the same management component as that of services themselves, which is deployment manager.The deployment manager represents a service in terms of SGPs; as it was conceived, the SGP gives an integrated and unified representation for deployment/service issues of TINA services. The details of the SGP representation are under study.

4. No draft available at the time of writing.

A-3.4 Service Customization

Three essential aspects have to be taken into consideration for customization: the service settings, i.e., the specific features and their possible values to be offered to users, the usage constraints under which a service can be executed and the configuration requirements in terms of terminal and network access points needed to support the service. Depending on stakeholders these aspects have different meanings. In Figure A1-3-7 the main characteristics of a service are represented.

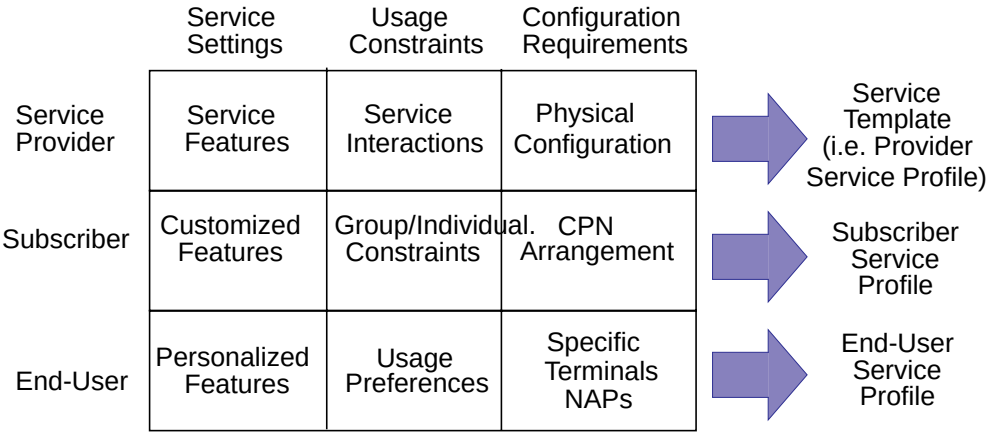


Figure A1-3-7. Customization of Services

A Service Provider can refine a service template by adding a set of service aspects or features. Moreover certain constraints on the execution of the service can hold (they are related to service interactions: a certain service cannot be executed in combination with other services or certain features are exclusive). A physical configuration can be required in order to provide a service (e.g., some terminal or a specific network). A Subscriber can accept a generic service offer and request a customization of certain features. A limited set of features of the service is accepted and particular values for them are agreed between the service provider and the subscriber. Restriction on the usage of the service can be imposed by the subscriber to individuals or group of users or on the service as a whole (availability during certain period of time). In addition, the service has to be tuned for the physical configuration and capabilities of the Customer Premises Network. Eventually each single user can choose values for specific service features, can define some preferences on the execution of the service and associate the service to a specific terminal or Network Access Point. Figure A1-3-8 shows overview of service customization.

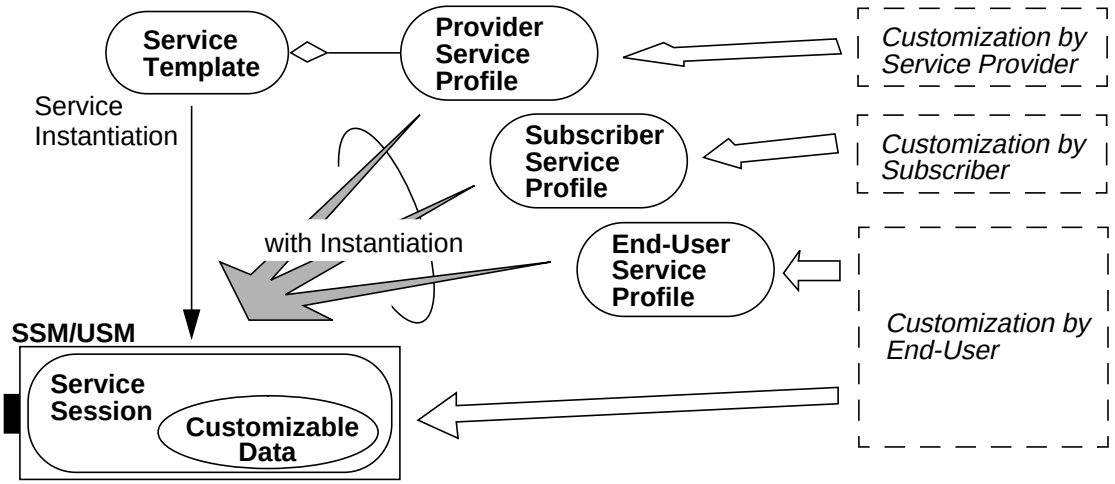


Figure A1-3-8. Service Customization overview

Customization by Service Provider

Service Provider can customize characteristic of service by modifying of service template. The modification of service template affects to the service which will be instantiated after modification. Namely, the modification does not affect to the service already instantiated. This customization is effective, for example, through the domain that this template is effective (i.e. Service Domain). Therefore, several different service templates, with different customization, for one service may exist in a service provider domain.

Customization by Subscriber

Subscriber can customize characteristic of service by modifying of subscriber service profile. The modification of subscriber service profile affects to the service which will be instantiated after modification. Namely, the modification does not affect to the service already instantiated. This customization is effective to the end-users to whom the subscriber is responsible[5]. A modification on the service is done for a service assignment group. Subscriber may have several different subscriber service profiles for one service, and those may be different service assignment groups for one service.

A-3.4.1 Customization by End-User

There are two ways to customize characteristic of service by end-user. One is to modify end-user service profile which is similar to the subscriber customization. Another is to modify the customizable data which has bundled to a service instance.

The modification of end-user service profile affects to the service which will be instantiated after modification. Namely, the modification does not affect to the service already instantiated. As a modification on the service is done by/for individual end-user, this customization is effective to the

5. Correctly, this customization is effective to the end-users, those who belong to service assignment group that the customization is applied to.

individual end-user. This modification has nothing to do with other end-user's services. The modification of service template, subscriber service profile and end-user service profile is categorized into static customization, that is effective before and/or at the service instantiation.

On the other hand, a modification of customizable data in a service instance directly affects that service instance itself. This modification does not affect to any other service instances nor end-user service profile. As a modification on the service is done by/for individual service instance, this customization is effective to the individual service instance. This modification has nothing to do with other end-user's services. The modification of customizable data in a service instance is categorized into dynamic customization, that is effective after the service instantiation.

A-3.4.2 Customization by Anonymous-User

According to the user and provider paradigm, user is categorized into three, subscriber, end-user and anonymous-user. Previous sub chapters explain the customization by (service) provider, by subscriber and by end-user. The remaining, customization by anonymous-user, is same as dynamic customization done by end-user[6]. Since anonymous-user does not have subscription contractual relationship with service provider, anonymous-user only customize service instances themselves that the anonymous-user currently using. There in no way to save customized data as profile, since the use of service and interaction to service provider are anonymous. This customization is effective only in this anonymous access or only in one service session. Therefore, the service provider customization which is done as service template is applied to the service that anonymous-user invoked, and service instance customization by anonymous-user may happen.

A-3.5 Management Context Information Model

Figure A1-3-9 illustrates the information model of the management context. Each individual FCAPS MgmtCtxt is envisioned as a sub-class of the top-level, MgmtCtxt class. Although the top-level MgmtCtxt class is expected to be the super-class of all the variants of FCAPS MgmtCtxts, it still awaits further studies in each management area to specify more details of the MgmtCtxt information model itself, as it has been observed that the information contents of the MgmtCtxt (e. g. accounting MgmtCtxt) tends to be quite specific to the management area it handles. In other words, what is most likely to be common properties across all types of management contexts are not specific to any management function, rather in its relationship to its associated service session (binding), and to the context management in view of the GCCM context configuration/management service.

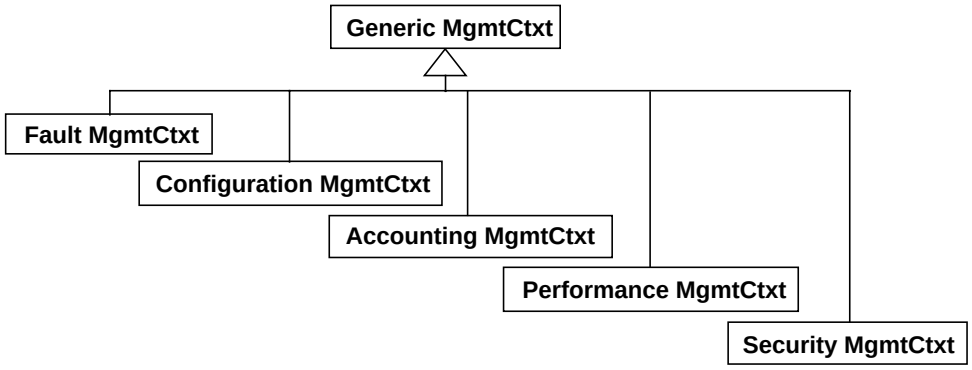


Figure A1-3-9. Inheritance hierarchy of Management Context

6. It may be possible to apply static customization for anonymous-user, as (an anonymous) user profile that is good through only in that anonymous access duration, but once logout then it will be purged.

Figure A1-3-10 shows aggregation relationship of management context

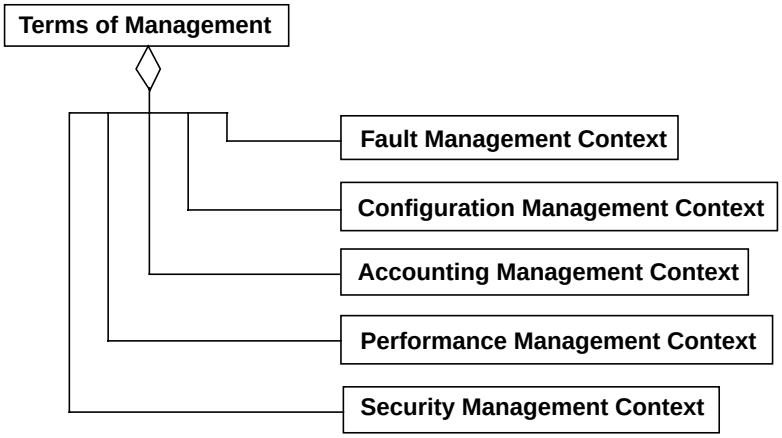


Figure A1-3-10. Relation of Management Context

A-3.6 The General Context Configuration Management Service

General context configuration management (GCCM) is positioned as a usage-assistance service in TINA service architecture. As such, GCCM service should be available across the service level reference points, including the Ret reference point. The objectives of the GCCM service are:

- **Semantics independent**: the GCCM service is semantics independent, which means that GCCM service is available for any type of context, but the semantics of the context needs to be interpreted by other semantic-dependent components used by the GCCM.
- **Negotiation**: the GCCM offers necessary and sufficient negotiation capability for general context management.

Figure A1-3-11 illustrates the computational model of GCCM.

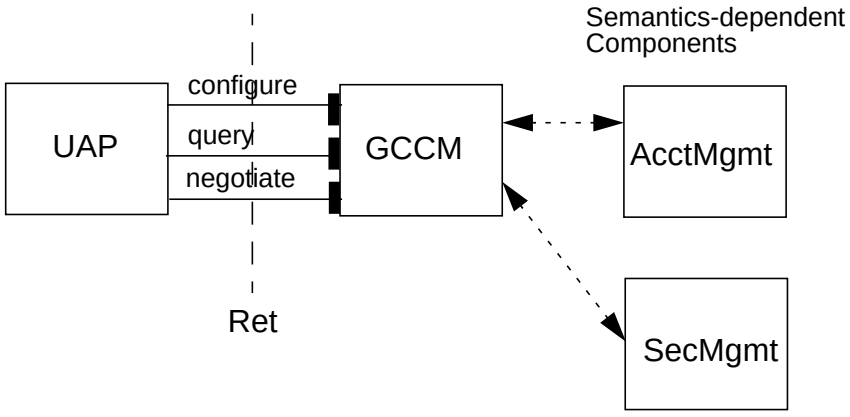


Figure A1-3-11. GCCM Computational Model

The GCCM object provides three interfaces:

- **Configure**: this interface is used when the UAP is allowed to set a context without negotiation, or when UAP asks the GCCM to set-up a pre-defined default context. It can be used if a user is invited to a session, or if plug-and-play is preferred and negotiation is not possible.
- **Query**: this interface is used when UAP asks GCCM for the current contents of the contexts, or the life cycle stage (live, dead, when modified, etc.) of the contexts.
- **Negotiate**: this interface is used when UAP and GCCM must negotiate the context.

Although the semantic part of the context is not considered here, three distinctive categories can be recognized, as they are evolutionary steps of the general context concept.

- **Static**: the context consists of a fixed set of attribute-value pairs. The scheme of the context is thus statically fixed. Negotiation has a clear meaning in this case, since the goal of the negotiation is a set of values agreeable to both sides.
- **Dynamic**: the context is still mostly made of the attribute-value pairs, but the scheme is dynamically expandable, i. e. it is allowed to add a new attribute-pair on the fly.[7] The negotiation process can be more complex, and more intelligence on both sides (UAP and GCCM) is necessary.
- **Executable**: the context is not only dynamic, but now it can be executable in the provider domain. It may resemble a program more than data, like a shell script or Java script. The context may contain an intelligent mobile agent, which is capable of negotiating for system resources directly with other service components in the provider domain.

Though the contexts conceived so far in TINA service architecture are mostly at the first two levels, it is envisaged that the mobile or intelligent agent type configuration management will be more prevalent in the future.

7. For example, CORBA Context object (CORBA 2.0, section 4.5) belongs to this category.

A-3.7 Service Lifecycle Management

A-3.7.1 Information Viewpoint

A-3.7.1.1 Service Design and Development

Figure A1-3-12 shows information model for TINA service from the viewpoint of service provisioning aspects, i.e. need and construction of service life cycle.

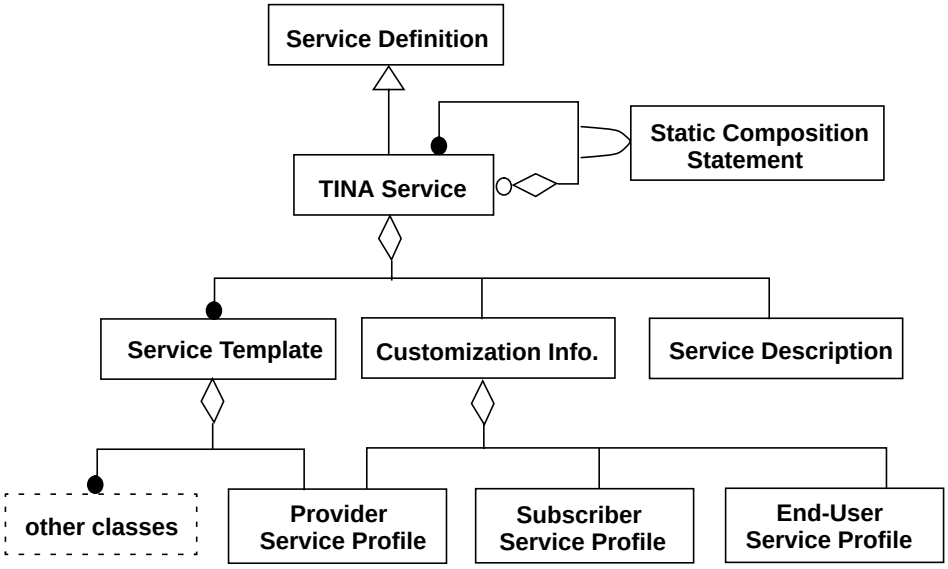


Figure A1-3-12. Service Model from service provisioning aspects

A-3.7.1.2 Deployment and Withdrawal

Figure A1-3-13 shows information model for TINA service from the viewpoint of service deployment and withdrawal aspects of service life cycle.

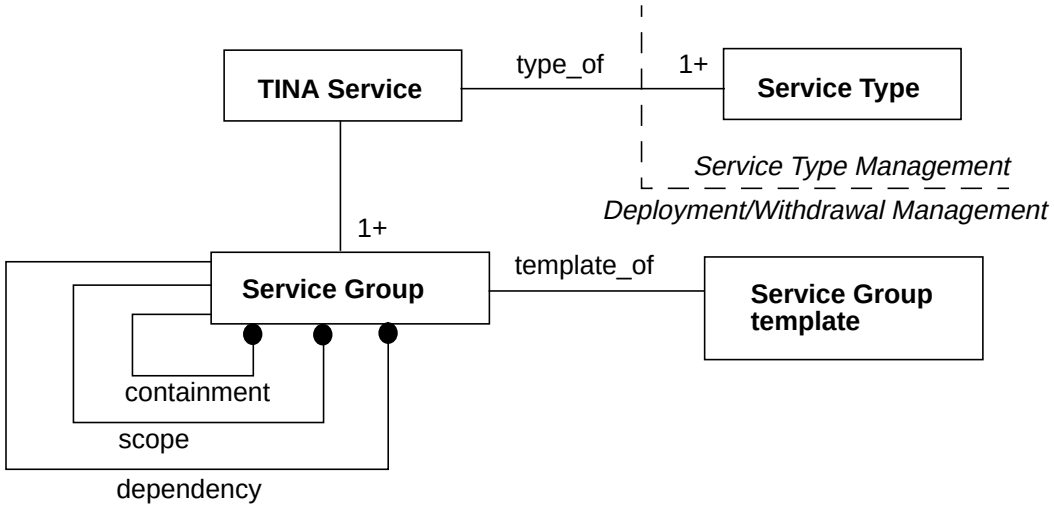


Figure A1-3-13. Service Model from Deployment and Withdrawal aspects

A-3.7.1.3 Utilization

Figure A1-3-14 shows information model for TINA service from the viewpoint of instance management aspects in utilization phase of service life cycle.

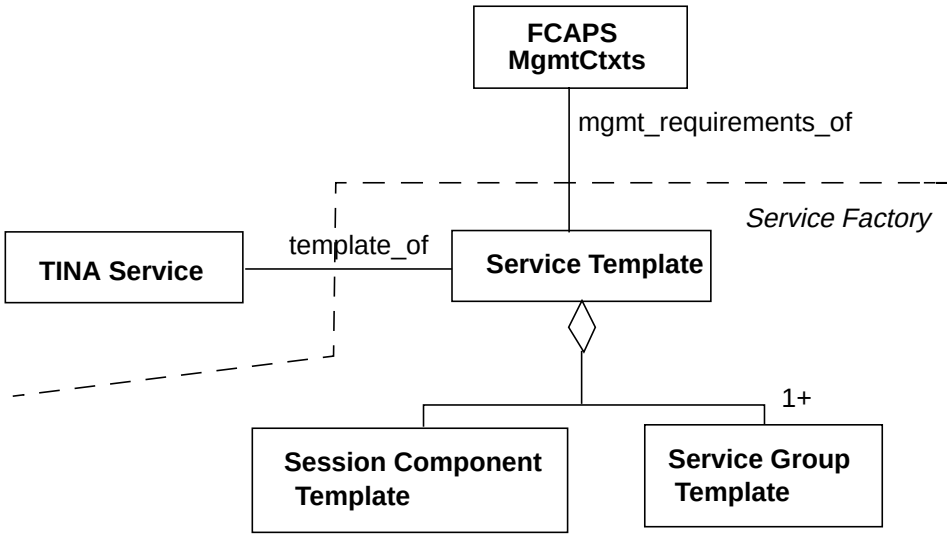


Figure 3-14. Service model from Service Instance Life Cycle Management aspects

A-3.8 User Lifecycle Management

A-3.8.1 Information Viewpoint

Figure A1-3-15 shows end-user and subscription related information model. In this figure, the right side which is service type management in a service life cycle management is described in previous section, and is showed only for an explanation.

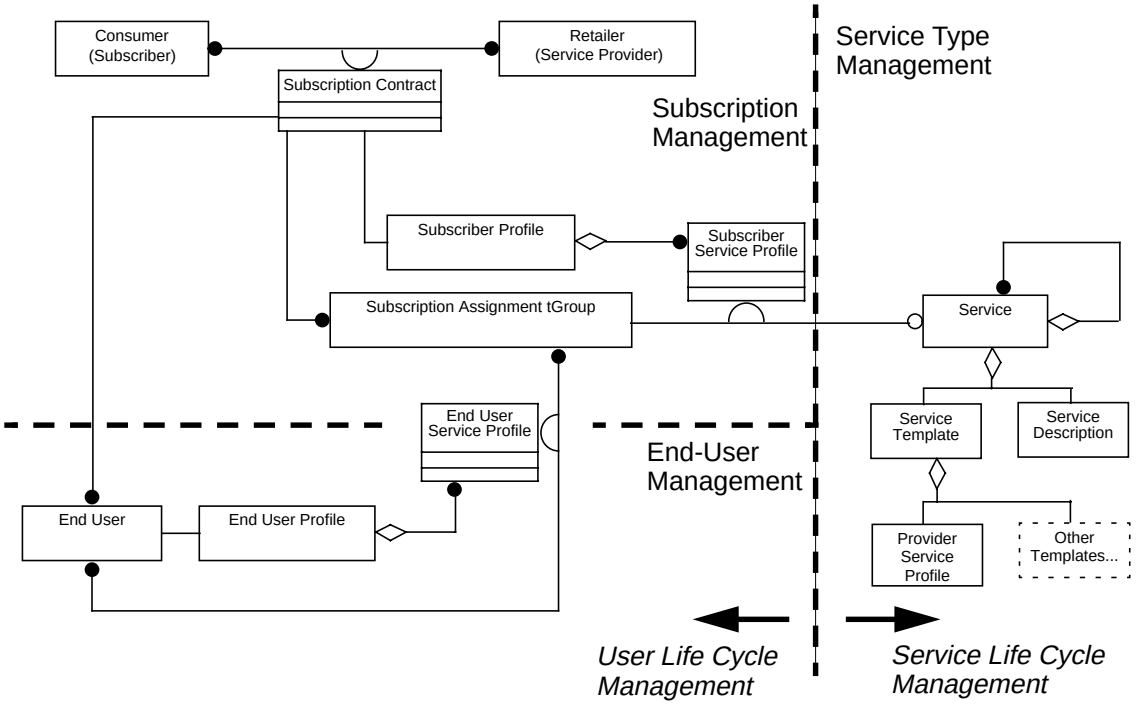


Figure A1-3-15. Information model from the user life cycle management

A-3.8.2 Subscription Management

The subscriber life cycle interacts explicitly with the service life-cycle in the phase of subscription and its termination, and this interaction is relevant to subscriber control and operation in the service life-cycle. The *Subscription Manager* is a management component in this fragment which is responsible for the control, management, and maintenance of subscription with the following components.

- **Subscription contract**: Subscription contract is an information object which expresses the generic concept of “subscription” and is a relationship established between consumer and retailer. Once a subscription relationship is established, a consumer is recognized as a subscriber. A consumer can subscribe to multiple retailers, and of course a retailer can have multiple consumers. One subscription contract expresses one relationship between a consumer and a retailer, i.e. it does not cover one to multiple contractual relationship.
- **Subscriber Profile**: Subscriber Profile is an information object which contains all relevant information regarding a subscriber. Both the subscription contract and subscriber profile contain subscription related information. The subscriber profile contains the service relat-

ed subscription information, such as service profile for subscriber, while the subscription contract contains the contractual information. Subscription profile aggregates subscriber service profiles.

- **Subscriber Service Profile**: Subscriber Service Profile is an information object which contains service related information, such as customization information and so on, relating to subscriber. Subscriber and end-user service profiles are distinct: the subscriber service profile affects all end-users who are associated with a subscriber. An end-user service profile is specific to one end-user.
- **Service Assignment Group (SAG)**: SAG is an information object which binds a service to end-users. A subscriber is associated with one or more SAGs. A SAG is assigned to a service in a subscriber service profile. Several SAGs are able to be assigned to a service together with different subscriber service profiles. A SAG may have multiple end-users, and a end-user may belong to multiple SAGs. All members of the SAG can use services assigned to that SAG.

A-3.8.3 End-User Management

The *User Manager* and User Agent is a management component in this fragment which is responsible for the control, management, and maintenance of end-user in association with the following components:

- **End-user profile**: End-user profile is an information object which contains all relevant information regarding a end-user. The end-user profile aggregates four objects, namely, registration, session description, usage context and end-user service profile.
- **End-user service profile**: End-user service profile is an information object which contains service related information, such as customization information and so on, relating to an end-user. An end-user may use a service which has been configured by a service provider template that affects the retailer domain, customized by a subscriber service profile that affects the subscriber's domain and customized by their personal end-user service profile that affects only that individual's service set-up.

A-3.8.4 Computational Viewpoint

Computational objects related to subscription management should support for subscribers to allow, with certain constraints, users to access services. The Service Architecture defines generic capabilities needed to manage, from the subscription point of view, different types of services. The computational objects related to Subscription include (Figure A1-3-16):

- Subscription Agent
- Subscription Manager
- Service Type Manager
- User Agent
- User Manager

Subscription Agents and Subscription Manager provide service-independent capabilities. However, reflecting a variety of services provided by service providers, Subscription Agent and related information objects need to be specialized to support particular services.

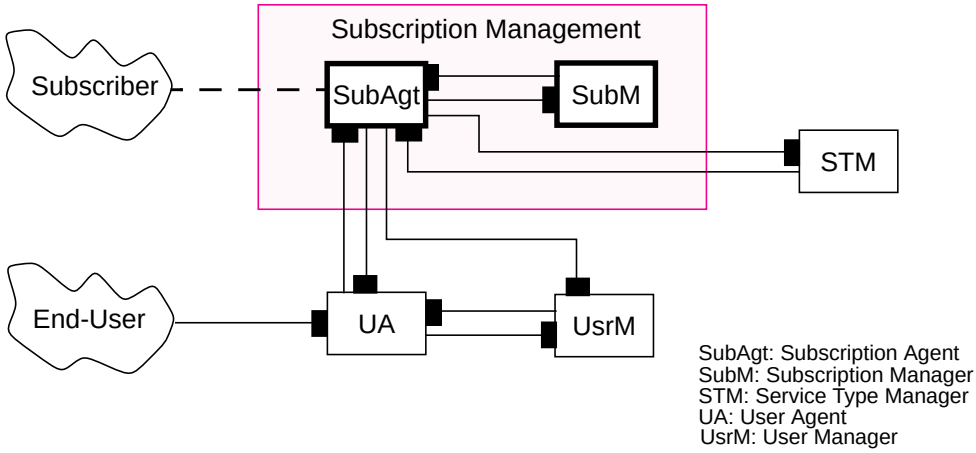


Figure 3-16. Subscription management related computational objects

Table 3-1. Mapping between subscription management related objects

Information Objects	Computational Object
Subscriber	Subscriber Manager
Subscription Contract Subscription Assignment Group Subscriber Profile	Subscription Agent
Service Template	Service Type Manager
End-user Profile	User Agent

A-3.8.4.1 Subscription Agent[8]

A Subscription Agent is a representation of subscriber (or subscription contractual relationship[9]) which is described in user/provider paradigm. A Subscription Agent is responsible for one subscription relationship between subscriber and service provider. A Subscription Agent is a contact point for accessing subscription information from User Agents. A Subscription Agent sends appropriate subscription information (e.g., a list of available services for the user and a set of constraints posed by a service provider and a subscriber to invoke a service) to a UA in reply to its request. Subscription Agent does not offer direct operational interface to the users, namely PA or UAP in user domain.

8. Subscription Register object was deleted and its functionalities was covered by Subscription Agent CO.
9. Subscriber may or may not be a “person“. It may be an administrative organization such as company, together with contact name and so on.

A-3.8.4.2 Subscription Manager

A Subscription Manager, which belongs to Subscription Management area, is responsible for subscribers in its management domain. In subscription management aspects, it provides subscriber lifecycle management operations, such as create provider subscription, delete provider subscription and so on, to the appropriate entities.

A-3.8.4.3 Service Type Manager[10]

A Service Type Manager, which belongs to Service Type Management area, is responsible for handling Service Templates (information objects) that represent service capabilities provided by the service provider. In subscription management area, it provides enquiry operations, such as list of all services that subscriber can subscribe, obtain customization information and so on, to the corresponding Subscription Agent.

3.8.4.4 User Agent

The User Agent, which is defined in Section 7.2 as a access session related computational object, represents a user.

3.8.4.5 User Manager

A User Manager, which belongs to User Management area, is responsible for users in its management domain. In subscription management aspects, it provides user lifecycle management operations, such as create end-user, delete end-user and so on, to the corresponding Subscription Agent.

10. Service Template Handler object was replace to this Service Type Manager CO and is separated from subscription area. Then, A Service Type Manager is defined to offer services to the Subscription Agent COs.

A-4 Examples for Composition and Federation

Example scenarios for composition and federation:

- A service starts another service and acts in the usage provider role,
- Progressive setup of federation Inviting a user in another domain to join an existing service session,
  - Sending an invitation
  - Joining a federated service,
- Joining an already federated service session.

Note that the scenarios are examples and that they assume all the operations are successfully completed (no error, no fault, and no rejection) for simplicity.

A-4.1 Composition: Usage Provider Paradigm

This example shows one service using another to create a composed service.

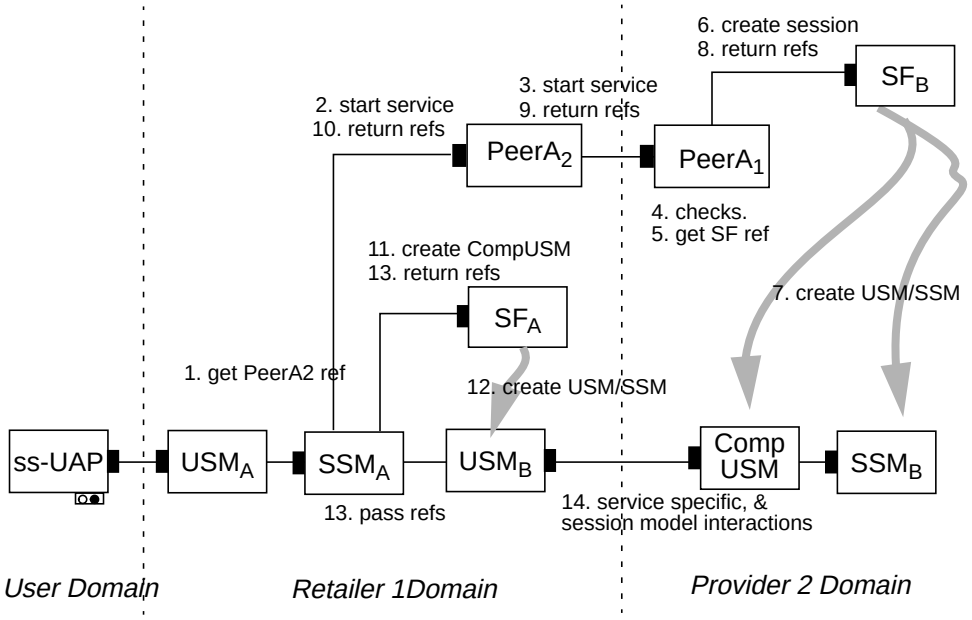


Figure A1-4-1. Starting a new service session.

Preconditions:

A service session exists in retailer domain 1. To fulfill a user request, it needs to make use another service of type B. Provider 2 supports service type B. This service B takes a usage party role, as it can control the shared service to make use of common facilities to export its services to end-users.

1. SSM_A, the service A service session manager, locates the domain access component, $PeerA_2$ of a provider offering service B (a PA or Peer Agent).
2. SSM_A requests a new service session of service type B, from $PeerA_2$. (The SSM passes information, including required session models, usage role - party in this case, feature sets supported, management context information, and interface references).

3. PeerA $_2$ requests to start a new service session of the service type (B), to PeerA $_1$, the domain access component representing the Retailer in the provider's domain (a UA or Peer Agent). It also passes the requested model, role, feature sets, management context, and other service information
4. $PeerA_1$ may perform some actions, which are not prescribed by TINA, before continuing. For example, the $PeerA_1$ checks the new session request against the contracts between these domains, to verify that the request is valid. Other decisions may also be taken. $PeerA_1$ raises an exception to the $PeerA_2$, if $PeerA_1$ declines to start the service session.
5. PeerA $_1$ gets a reference to a service factory which can create service session components for the service type (B). [1]
6. UA requests a new session of the service type (B), supporting the required session model, usage role and feature set is created by the Service Factory.
7. Service Factory creates an SSM(SSM_B) and a composing usage session manager (CompUSM), and initializes them.
8. Service Factory returns interface references of the CompUSM and SSM_B, to PeerA $_1$.
9. UA returns references of CompUSM and SSM_B to the PeerA $_2$.
10. PeerA $_2$ returns references of CompUSM and SSM_B to SSM_A.
11. SSM_A requests that its SF create a USM, supporting the same feature sets, service model as requested for service B, exporting the provider role as usual.
12. The SF creates and initializes the USM_B to support interactions between the services
13. The SF returns the USM_B interface references to the SSM_A
14. If necessary, the SSM_A passes the CompUSM and SSM_B references to the USM_B
15. The CompUSM and USM_B can now interact using specified session model and feature set interactions. Some interactions may be necessary before the provider service is fully integrated in to the other service. The service session B takes the usage party role, and can use service A much like an end-user.

While this may sound unusual, there are cases where this kind of behaviour is useful. Note that in general, a combination of session model, feature set and exported role determines the usage session management component to be created.

A-4.2 Progressive federation

In this example, the federation is setup progressively, rather than waiting for the end user to accept an invitation and request to join before commencing federation.

1. The UA may use a location service to find an appropriate service factory, or some other means. The UA may also provide other information to scope the search for the service factory, such as terminal configuration information. Other means include: the subscription information could potentially contain an interface reference to the service factory to use.

A-4.2.1 Sending an invitation and initiating the a new session

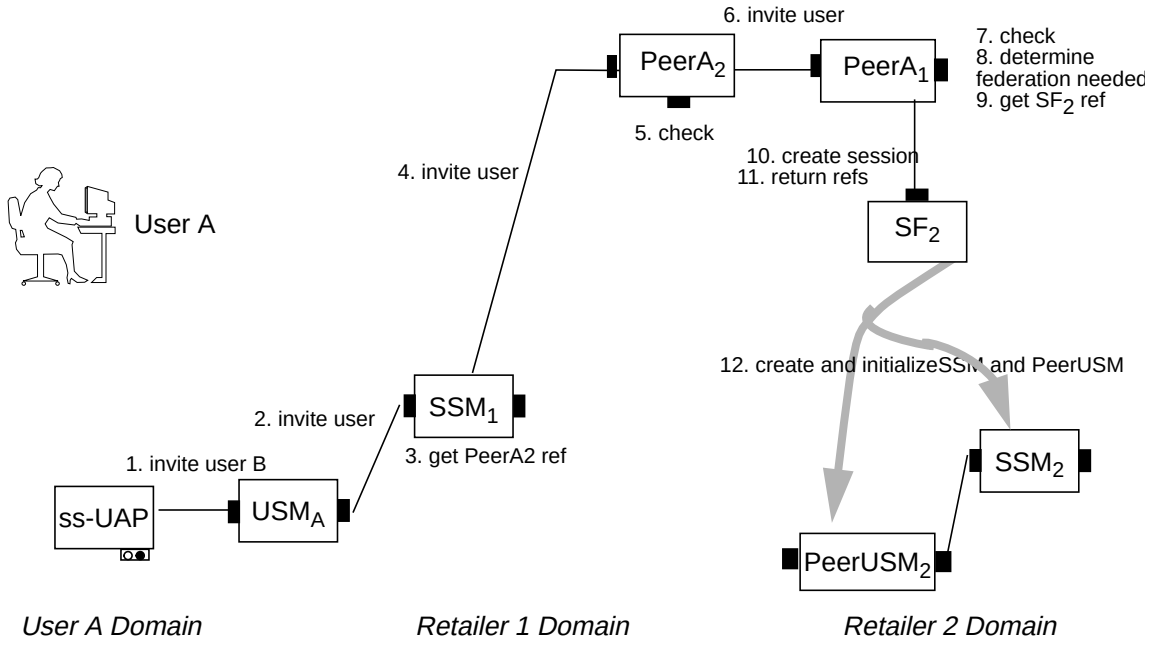


Figure A1-4-2. Joining an existing service session.

In this case, the session either explicitly sends an invitation to a domain to start a federation, or the access session in the other domain determines to start a federated session on receiving this invitation.

1. User A uses UAP to invite a user (invitee) to join a session. (User supplies the user name of the invitee, or a user defined alias which can be resolved by the inviter's UA). The UAP requests[2] the USM to invite user B to join the session.
2. USM requests the SSM to invite a user to join the session.
3. SSM gets a reference to an invitation interface that can act for user B - The SSM need not be aware that this is in fact an interface of the Peer Agent acting for retailer 2, $PeerA_2$.
4. SSM sends an invitation using the invitation interface of $PeerA_2$.
5. $PeerA_2$ may perform some actions, which are not prescribed by TINA, before continuing. For example, it may check that this is a user that it knows about or that the invitation conforms to the federation contract (e.g. for a known service type). The policy will then determine the $PeerA_2$ actions, and interactions with other objects. $PeerA_2$ may raise an exception to the SSM, if the $PeerA_2$ declines to deliver the invitation.
6. $PeerA_2$ sends the invitation to $PeerA_1$, the peer agent representing this access session in Retailer 2's domain, using its invitation interface.
7. $PeerA_1$ may also check the validity of the invitation, raising an exception with $PeerA_2$ if it needs to refuse the invitation request.

2. This request can be made using the Multiparty Feature Set interface on the USM, if the TINA session model is being used; or on a service specific interface.

8. $PeerA_1$ realizes that a federation is required. It may check federation agreements at this point before proceeding.
9. $PeerA_1$ acquires a suitable service factory reference (to SF_2)
10. $PeerA_1$ requests SF_2 to create a service session of the required type and to support user B in a normal usage party role and to support federated session 1 in a usage peer role.
11. The SF_2 creates and initializes the requested session components.

A-4.2.2 Sending a join federation request

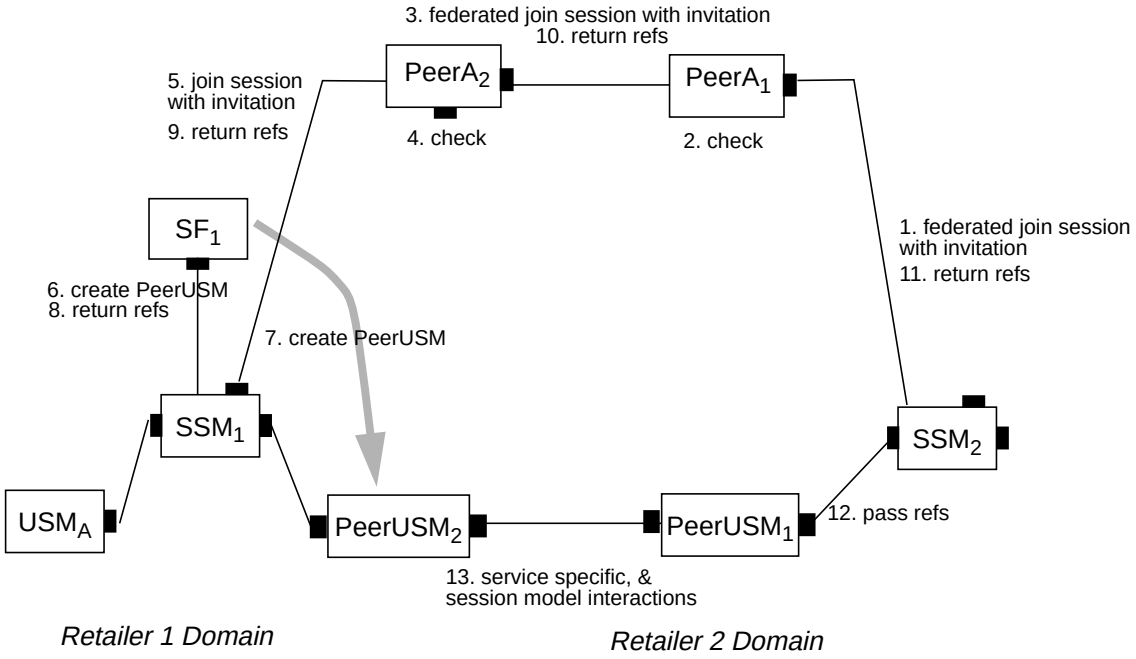


Figure A1-4-3. Joining an existing service session.

We assume that the newly created session has the original invitation. In response to this invitation it generates a federation join request with invitation. This join request indicates that the session wishes to

1. SSM_2 is initialized with the initial invitation. It realizes it needs to send a join request to $PeerA_1$ to create the federation.
2. $PeerA_1$ checks the join request is for a valid invitation. It may also check context information and a context negotiation could be commenced at this point.
3. $PeerA_1$ sends the join request to $PeerA_2$
4. $PeerA_2$ checks the request, determines if the invitation is valid and locates SSM_1
5. $PeerA_2$ sends the federated join request to SSM_1
6. SSM_1 responds by creating a federating usage session manager to SF_1.
7. SF_1 creates the required component, $PeerUSM_1$, and initializes it
8. SF_1 returns the interface reference to $PeerUSM_1$ to SSM_1
9. SSM_1 acquires the exported interfaces of $PeerUSM_1$ and returns them to $PeerA_2$

10. $PeerA_2$ returns the exported interfaces to $PeerA_1$
11. $PeerA_1$ returns the exported interfaces to SSM_2
12. SSM_2 passes on the exported interfaces to $PeerUSM_2$
13. Interactions between the two PeerUSMs may now commence to support service specific and session model interactions. Notice however, that the actual user B has not yet necessarily joined the session

A-4.2.3 Sending on the invitation to the end user.

An invitation needs to be sent to user B. This would follow steps 4 to 8 of the simple invitation case in Section 7.5.2.

A-4.2.4 User B joins, completing the federation.

The federation is only complete when User B actually joins the session. This scenario follows the simple join case in Section 7.5.3. The only extension is that service session 2 would update service session 1 to inform it that user B had joined, using the session model or service specific interactions.

If the invitation to user B is not responded to, this could result in the deletion of service session 2, ending the federation.

A-4.3 Joining an already federated service session

This example shows a user C being invited to join and joining an existing federated session. The invitation comes from user A in another federated domain. User C is assumed to be in an access session with the provider, and to have a valid subscription to the service. The service session related UAP is assumed to be present on user C's terminal.

Preconditions:

There exists a federated service session established involving user A in retailer 1's domain and user B in retailer 2's domain. Both retailer domains are federated (federated access and service sessions are established between them). This is the situation after the previous example (or after the federation example in the main body). User A wishes to invite user C to join this service session. User A is 'active' in the service session, i.e. they have not suspended their participation. User C belongs to retailer 2's domain. This example also assumes that an access session does exist for user C.

A-4.3.1 Sending an Invitation to a User in another Domain.

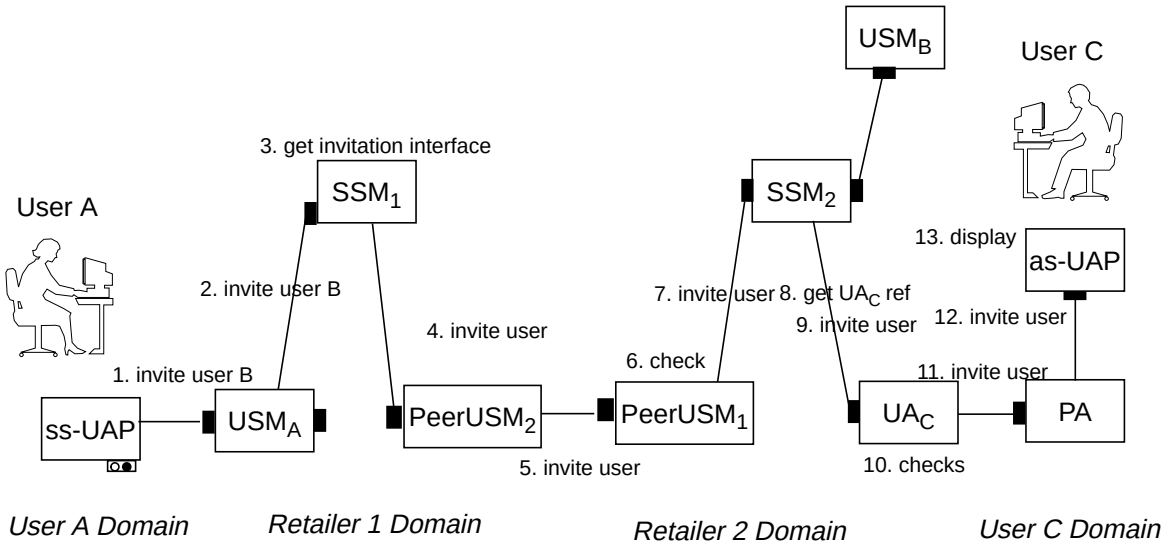


Figure A1-4-4. Inviting user C to an existing federated service session.

Only steps 3 to 9 are different from the ones in example in Section 7.4.8. These steps are described below.

3. The SSM gets a reference to an invitation interface that can act for user C[3], $PeerUSM_2$.
4. SSM request $PeerUSM_2$ to invite user C in retailer 2 domain.
5. $PeerUSM_2$ forwards the invitation to $PeerUSM_1$.
6. $PeerUSM_1$ may performs some actions, which are not prescribed by TINA, before continuing. For example, it may check that this invitation is acceptable and conforms with the terms of the federation contract (federation context). $PeerUSM_1$ may raise an exception to $PeerUSM_2$ in case it declines to deliver the invitation.
7. $PeerUSM_1$ makes a request to invite user C to the SSM. From here on the behaviour is the same as in the case of an invitation from a local user. SSM2 may not be aware of the fact that the invitation is coming from a user that does not belong to its domain.
8. SSM_2 gets the invitation interface reference of user C's UA.
9. SSM_2 sends the invitation to UA_C.

The invitation to join the service session has been delivered to user B's UA, PA, and is displayed by the UAP. The invitation contains sufficient information for the UA to locate the service session, and allow the user to join it. Next paragraphs describe the join with invitation.

3. We are assuming that SSM_1, by some means, realize that user C belongs to retailer 2 domain with which it has a federation relationship, supported by $PeerUSM_2$. A location service might not suffice for that. Additional service logic or supporting services might be required in the SSM.

A-4.3.2 A User Joins a Previously Federated Session.

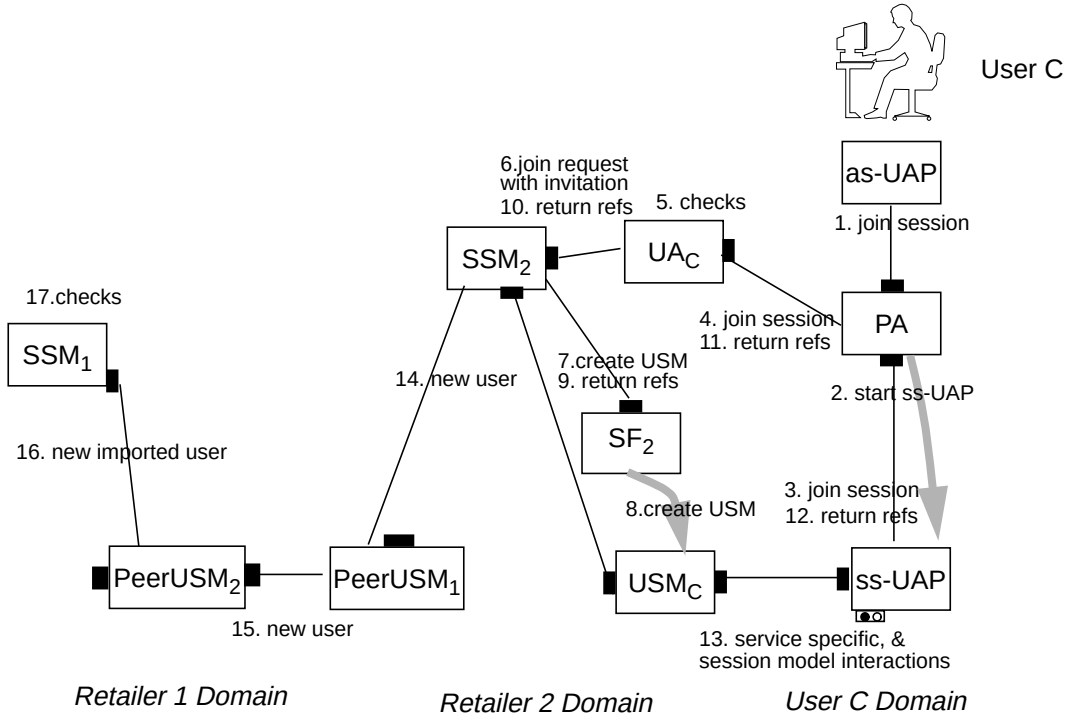


Figure A1-4-5. Joining an existing federated service session.

The first 13 steps are the same as in the example in Section 7.4.5]. Only the interactions required to notify the join of user C to the federated session are described.

14. SSM_2 has kept track of the origin of the invitation (invitation information like invitation id and the inviting member is part of the join with invitation request) and informs $PeerUSM_1$, representing session 1, of the change in the session 2's SSG.
15. $PeerUSM_1$ forwards the indication (a new user joined with invitation) to $PeerUSM_2$.
16. $PeerUSM_2$, in its turn, request SSM_1 to incorporate the imported user to its session.
17. SSM_1 may perform some actions before modifying the SSG. For example, it may verify that the user was invited (some identification may have been passed with the invitation for this confirmation).

The evolution of the service session graph along this example is shown in Section A-4.4.

A-4.4 Evolution of the service session graph during a federation session

This examples are based on examples in Section 7.4.8 and in Section A-4.3. Use Figure A-7-9, page A-122 and Figure A-4-4, page A-50 to follow the descriptions below.

A-4.4.1 The Session Graph and Invitations.

To invite a user means, in terms of session graph operations, to request the creation of new party in the session graph. If the user belongs to another domain, some other operations on the session graph may be needed. This example intends to illustrate the service session graph support for federation (and by extension to composition). It refers to the invitation scenarios in Section 7.4.8.2 and Section A-4.3.1.

The invitation phase itself requires no modification of the SSG. The SSG is not modified until the party actually joins. However, it may be queried during the processing of the invitation in the SSM to check whether retailer 2 is already federated (a peer session member may represent it in the SSG).

A-4.4.2 Session Graph and Joining a Federated Service Session

Now let us consider the join scenario of Section 7.4.8.2. In this scenario, a user responds to an invitation from a service session in another retailer domain. Figure A-4-4, page A-50 shows how the service session graph in both domains evolves during that scenario[4]:

1. Before the invitation, session 1's SSG contains only party A. After step 12, a peer is added, representing session 2.
2. When in step 17 the SF in retailer 2's domain creates the SSM, this is initialized with a session graph that contains a party (user B), a peer (representing the federated session 1) and a composing session relationship between them indicating that partyB is exported to peer1 (ShSR). The decision of creating the ShSR (exporting the party) is derived from the terms agreed in the federated access session or from internal federation policies. In case of a join with invitation in which the invitation is received from the federated session, the party is most probably shared.
3. SSM_2 makes an export indication to $PeerUSM_1$. In the first interaction with the $PeerUSM_2$ (step 18), $PeerUSM_1$ passes the export indication (add new imported user). The former forwards it to session 1's SSM. This causes the creation of a new party in the SSG (party B) and its association (through a subordinate SR) to the exporting peer.
4. As a response to this first interaction (step 18), $PeerUSM_2$ passes to $PeerUSM_1$ the parts of the session graph that session 1 decides to let session 2 know[5] (i.e., decides to export). Previously, session 1 needs to create a ShSR between the exported members (party A) and the importing peer (peer2). This causes the incorporation of party A to session 2's SSG and its association with the peer member representing session 1 (peer1), through a SubSR.

After these initial interactions between PeerUSMs, the federated service sessions work on two different, but consistent session graphs.

4. Steps in this section refer to the sequence of the example in Section 7.4.8.2, Figure 7-9
5. This decision can be based on the agreement reached in the federated access session or on internal federation policies. $PeerUSM_2$ may need to request this information to its SSM, SSM_1.

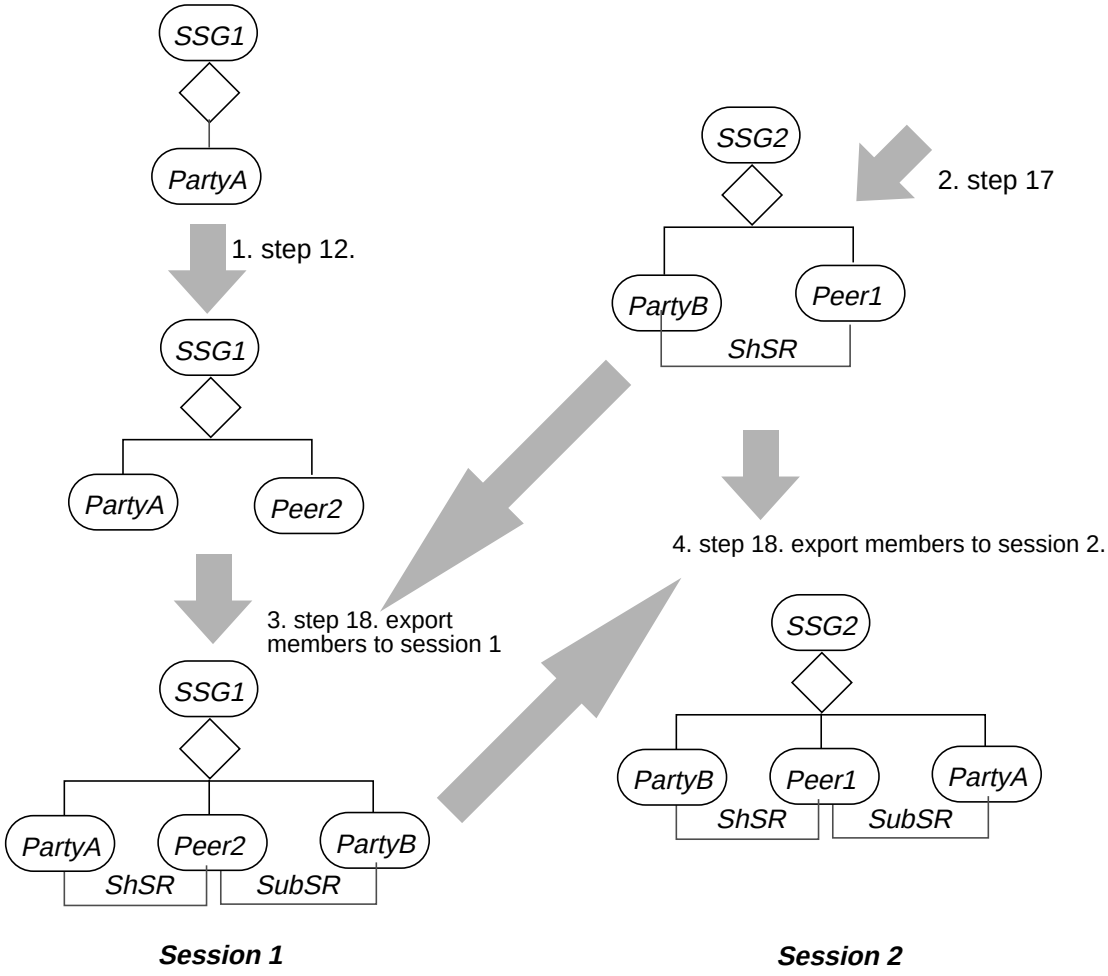


Figure A1-4-6. Service session graphs evolution in federation on joining example.

A-4.4.3 Inviting a user to join an already federated service session

Now let us consider the scenario where a federated session already exists, as described in Section A-4.3.2. As in the case above, no changes are made on the session graph during the invitation phase. The evolution of the SSG in the join phase is shown in Figure A-4-4, page A-50. The two main phases in the update of the session graphs are[6]:

1. After the USM for party C is created (step 9), Party C is incorporated in session 2's SSG. Once decided that the party needs to be exported[7] a ShSR is created from this party to the session it is exported to (represented by a peer, peer 1).
2. After receiving the indication to incorporate the new user (step 18), SSM_1 modifies its SSG adding the exported party (C) and a subordinate session relationship that associates it to the peer representing the exporting session (peer2).

6. Steps in this section are referring to the sequence in example in Section A-4.3.2, Figure A-4-4, page A-50
7. It is clear in a join with invitation when the invitation is received from the other session, but might not be so straightforward in case the user is joining under its own initiative or the invitation is done by a user in session 2.

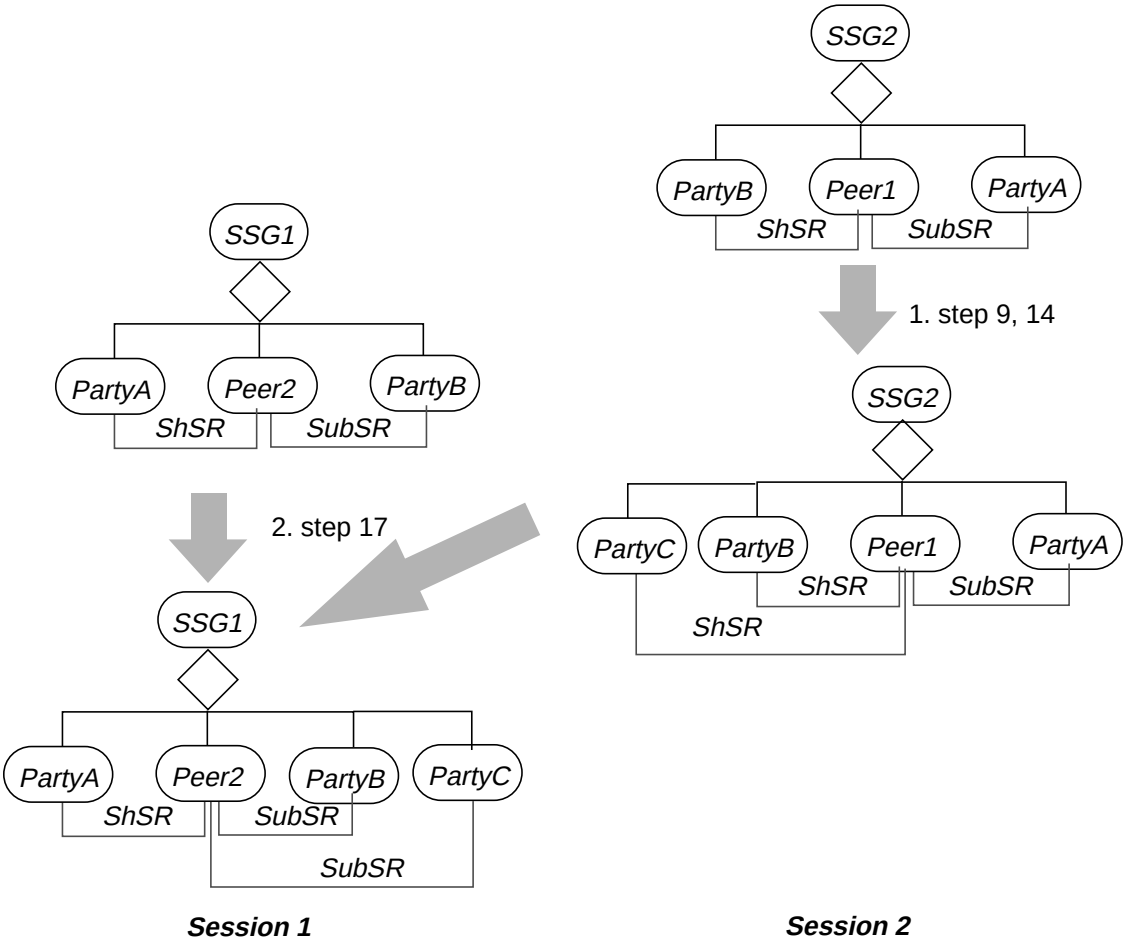


Figure A1-4-7. SSGs evolution on the “Joining an already federated session” example.

After this operations both SSGs are updated and consistent.

Cases in which some problem arises in the communication between both sessions or one of the sessions declines to export or import a member have not been considered in the federation examples. These situations can cause problems of inconsistencies[8] between SSGs. The recovery actions to solve them needs to be devised. These actions could be similar to the commit and rollback procedures in databases.

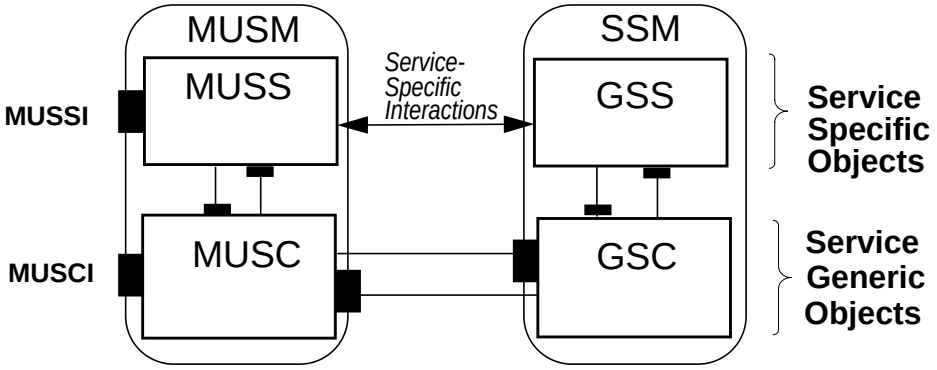
8. The rules to keep consistency between federated sessions have been defined in Section 6.4.6

A-5 Descriptive Refinements of Service Components

This section presents possible CO mappings (see Section 7.1 of the main body) for particular service components, and describes additional components for subscription aspects and to model service specific entities. It relies on the concepts of computational object (CO) and CO group, defined in [5].

A-5.1 A possible way to handle service specific behavior

The following separation will apply as well to the USM as the other subclasses of MUSM, like CompMS and PeerSM. Hence the generic terms MUSS and MUSC is chosen in this section. However each time the term 'MUSM computational object group' is used, the term should be replaced by the more concrete one 'USM computational object group' etc.



MUSM: MemberUsage Service Session Manager (or any subclass thereof like USM)
SSM: Service Session Manager
MUSS: Member Usage Service Segment
MUSC: Member Usage Session Control

GSS: Global Service Segment
GSS: Global Session Contro

MUSSI: MUSS Interface
MUSCI: MUSC Interface

Figure A1-5-1. CO mapping for the MUSM (and its subclasses, e.g., the USM) and SSM components onto generic and specific objects.

The MUSM and SSM service components *can* be mapped to computational objects and CO groups as shown in Figure A1-5-1. MUSM and SSM are CO groups composed each by two COs, one encapsulating the service-independent aspects, the other encapsulating the service-dependent aspects. This is one way of organizing the MUSM and SSM internals. Similar organization can be done for the UAP, creating a service specific part Service Session EndPoint (SSEP) and a Generic Session End Point (GSEP)

Other ways exists al well, e.g.,by allowing the SSM and MUSM (and it's subclasses) to have service specific behavior associated with a generic operation like invite party and create stream binding requests.

The MUSS and MUSC are, in the case of the USM, the constituents of the USM computational object group, and consequently they represent the user in the service session (local context / user context). The GSS and the GSC are the constituents of the SSM computational object group. They care for the global context of the service session (shared context) and handle the central control of the service session and its coordination.

This separation of the (M)USM will also be reflected at the UAP level by decomposition into the SSEP and GSEP. A picture of the computational objects involved at one user's side for both service specific and service generic segments is shown in Figure A1-5-2

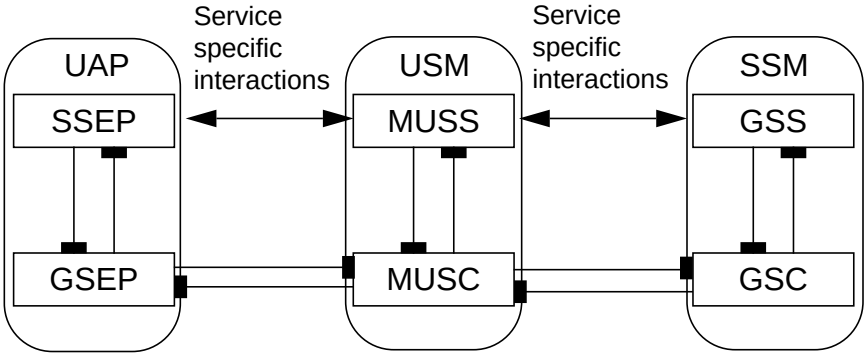


Figure A1-5-2. Computational model for the service session including the user domain based on a separation of generic and specific objects.

In supplement to the interfaces already described in this document, the MUSC and the GSC will offer to each other and to the GSEP, the SSEP, the USS and the GSS, interfaces for building the Service Session Graph (SSG). As stated in Section 6 of the main body, the SSG is an information model which models the configuration of the service session, and also the information exchanged between components in order to cause and notify changes in the configuration itself. This information, being service-independent, belongs to the service generic segment. The information exchange in terms of session graph can either be peer-to-peer within the service session segment (involving GSEP↔USC↔GSC) or based on a client-server interaction between the service segment (i.e. the USS and GSS COs) and the service session segment (i.e. the USC and GSC COs). The operations offered by the service session COs at their operational interfaces allow modifications of the SSG (and consequently of the service session configuration).

A-5.2 User domain refinements

In Section 7 of the Definition of Service Architecture the UAP has been considered as a component able to interact with other TINA components and offering interfaces to TINA. It is in fact considered as a TINA component. As software already exists today that can act as UAP in a user system (e.g. WWW browsers), it must be possible to use these existing (non-TINA) applications over the TINA environment. This chapter describes a way to guarantee this flexibility by clearly separate TINA components from non-TINA applications.

A-5.2.1 User application and Other Components in the User Domain

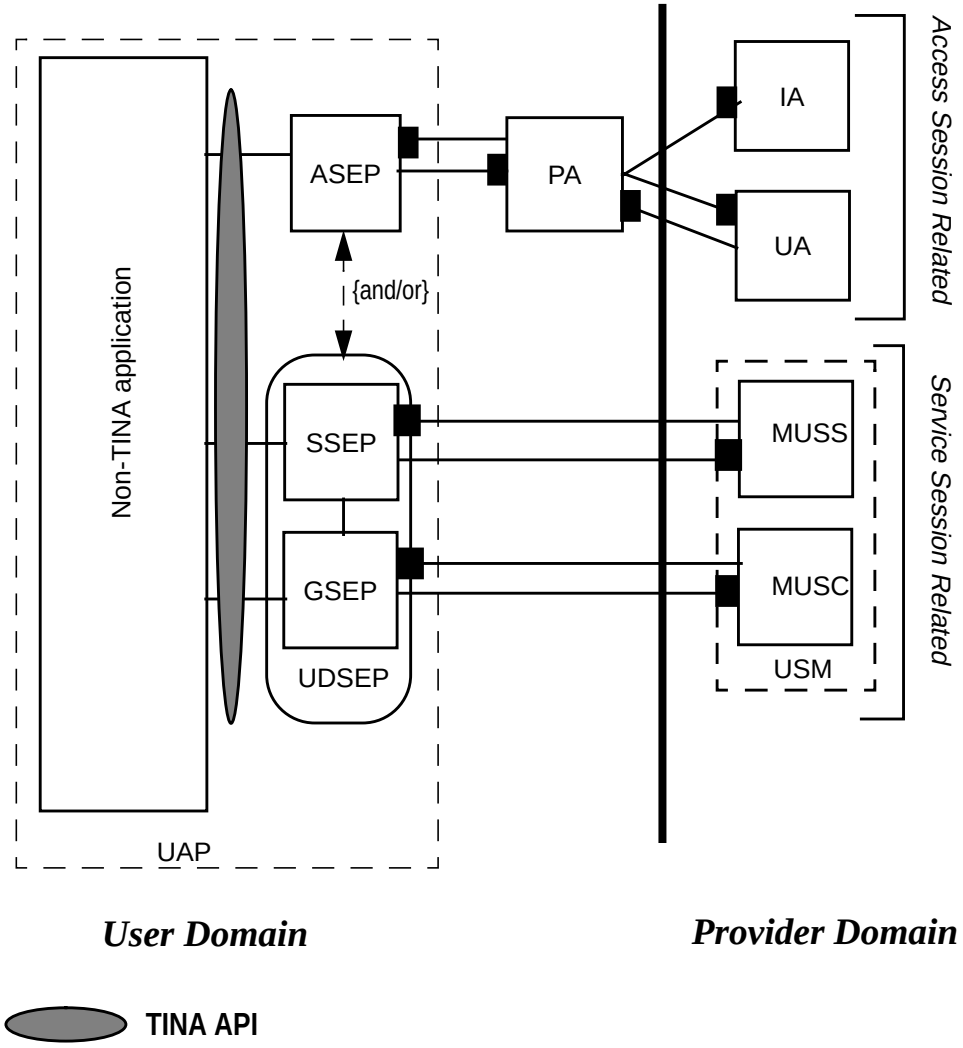


Figure A1-5-4. User Application Components for Non-TINA Applications.

The service component UAP is here further decomposed to allow for non-TINA applications. The term UAP here is therefore slightly different from the term UAP as used in the main body. For the same reason the object containing SSEP and GSEP is here called UDSEP, while it is called UAP in Section A-5.1

A-5.2.1.1 Access Session related components

These components are required during the access phase. The reader can referred to Section 7 of the Definition of Service Architecture for more information about the PA, IA and UA.

Non-TINA application The non-TINA application represents an existing application through which the user uses access to his retailer and/or uses his service. Indeed, the non-TINA application can be used only for access, usage or both. Non-TINA applications can be grouped by family depending on

the kind of API (or interface) they offer. For example, all browsers containing a Java virtual machine can be grouped together. In this case, the requirement for a TINA service component to use this non-TINA application family is to be wrapped in an applet.

Access Session End Point This component represents the access session end point in the user domain. It is dependent on one non-TINA application family and allows by this way the provider agent to be independent of this non-TINA application. It interacts with the provider agent by using the PA interface and by offering an interface to the provider agent.

A-5.2.1.2 Service Session related components

These components are required for the usage of the service. The reader can referred to Section 7 of the main body for more information about the USM (SSM). The UDSEP is explained below.

Non-TINA Application (as above)

User Domain Session End Point (UDSEP) The User Domain Session End Point (UDSEP)[1] is the counterpart of the USM (or SSM) in the user domain. We describe it as a CO group composed of a Specific Session End Point (SSEP) and a Generic Session End Point (GSEP) computational objects. It handles all service requests (specific and generic) from the retailer and interpret them in the context of the non-TINA application. In the other way, it receives requests from the non-TINA application and interpret them to call the USM or USM. The functionality of the UDSEP is not only to forward call (which is rather the functionality of a gateway), but also to interpret these calls in service specific way. The behavior of the UDSEP is indeed service specific even if it offers generic interface through the GSEP.

Specific Session End Point (SSEP) It is a computational object specific to the service and to the non-TINA application family.

Generic Session End Point (GSEP) The Generic Session End Point (GSEP) is a computational object supporting generic session control operations between a UAP and its USM/SSM. It may also support a session graph interface for each UAP interacting with a USM, if the USM/SSM uses a session graph to model the session participants, etc.

The GSEP supports the following generic session control capabilities:

- quitting the session.
- inviting other users to join the session.
- suspending the user's participation in the session, or the whole session.

The behavior of the capabilities depicted above is service specific and specific to a family of non-TINA application. The former means that depending on the service, a control can be ignored or interpreted in a specific way, the latter means that a control is presented to the user in a way that depends on the non-TINA application (an invite can generates a ring or a pop-up window depending on the non-TINA application).

A-5.2.1.3 Assumptions and clarifications

It is important to note that the different components depicted in Figure A1-5-4 can be supplied by different companies from different business. For example, the non-TINA application already exist today and is a totally "non-TINA" component, the GSEP (see Figure A1-5-4) can be supplied by TINA

1. AI already noted this is called UAP in the main body, but a renaming was necessary here in order to have the UAP as depicted in Figure A1-5-4

if it offers a session graph interface and then the SSEP can be supplied by a third party service provider who which to specialized the service end point by adding some service specific interfaces/ functionality.

It is also important to note that these different front-end component (ASEP, SSEP and GSEP) are not always deployed in the terminal but can also be deployed in the network. In this case, they are specific to a non-TINA protocol. Indeed, the computational model cares only about functionality but doesn't make any assumption about the deployment of these different components. It's an engineering problem. This deployment doesn't have any impact on the Ret reference point.

No assumptions are made about the way that the USM and UDSEP are designed. It can be by aggregation or inheritance but it is totally up to the designer. For the sake of simplicity, only the USM is depicted but a direct interaction between UDSEP and SSM is also possible.

A-5.2.1.4 Interface between non-TINA application and TINA user application components

The solution presented in this chapter is based on the fact that we are not able, for the moment, to define a TINA API for the usage of services. Nevertheless, an alternative to allow TINA components to interact with non-TINA application can be to considered the PA interface as the TINA access API and the session graph interface (when used) as the usage API. It means that the ASEP, SSEP and GSEP are totally dependant of the non-TINA application (or at least, a family of non-TINA application) and are used to wrapped the TINA API in such a way that it can interact with the non-TINA application. For instance, if the non-TINA application is a web browser which support a Java virtual machine, the TINA front-end components will be written in Java and wrapped in an applet. It means that these front-ends objects can also supply a gateway functionality. In the example above, for outgoing calls, the user events are caught by the Java events manager and translate into TINA ODL call (generic outgoing events). In the other way, these front-end objects translate generic incoming events (invite,...) in a way that can be sent to the non-TINA application (an invite event can corresponds to a pop-up window or a ring depending on the non-TINA application). Furthermore, these events are interpreted in the context of a service (which is a big difference comparing to the functionality of a gateway).

A-5.3 Additional Service Components

As defined in Section 7 (Definition of the Service Architecture), there can be additional components that have a longer life time than an access or service session, and are specific to a class of services or particular environments; they are termed Service Support Components (SSCs). Example of possible SSCs are described in this section.

A-5.3.1 Terminal Service Adaptor

The Terminal Service Adaptor (TSA) adapts the User Service Provider Interface (USPI) operations, which are offered by the USM, into (lower level) user interface actions.This object encapsulates the presentation (user interface) part of the application, adapted for a specific terminal, in terms of its graphic or multimedia technology (e.g. X-Windows etc.).

A TSA is be dependent both on the specific service and on the terminal. This component resides either in the provider domain or in the user domain. In the former case, the TSA will be instantiated in the provider domain, associated to a USM, by the service factory taking into account the information on the end-user terminal passed by the access session objects. In the latter case a large number of interchanged messages which usually implies a complex (graphical) user interface can be avoided. In order to realize session mobility, a new TSA has to be created when the user resumes the session, if needed.

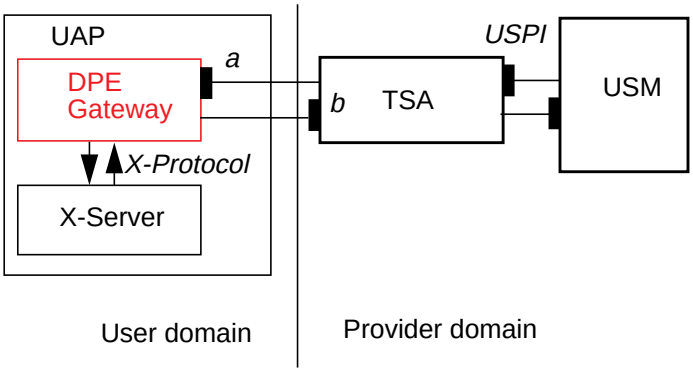


Figure A1-5-5. Example: TSA for X-Windows adaptation.

An example of this adaptor can be illustrated for X-Windows adaptation[2]. In this case, see Figure A1-5-5, the user is accessing the service using a workstation, acting, from the service point of view, as an X-terminal. The TSA would be an adapter of X-protocol primitives into USPI operations. The X-Protocol messages are interchanged between the TSA and a DPE gateway, which encapsulates them as DPE messages. The interface b will include a function like: x_event(), to encapsulate the events, and the interface *a* will include: x_command() to encapsulate the X- protocol actions. From the terminal point of view, the TSA is acting as an X-Client.

2. This example includes engineering/technology considerations in order to help the clarity of the use of TSAs.

A-6 TINA Common Services and Facilities

TINA is based on the CORBA DPE and then, relies on CORBA common object services and facilities. But, if these services/facilities defined by CORBA are deemed not satisfactory or if some others are deemed missing (may be, because specific to TINA), TINA has to define its own services/facilities. The purpose of this chapter is to identify these facilities specific to TINA and missing in CORBA.

The services and facilities defined in this chapter have been presented and discussed within the TINA-C core team over the past year. The Information Services document [25],which also detailed these services, has been available outside the core team. The stability of the services and their facilities is not complete. It is possible that the services may be changed, or more likely added to in the future.

Currently one issue is addressed:

A-6.1 Information Resource Storage

The goal of an information service is to locate, fetch and broadcast an information resource to users. The retailer in TINA is the first contact point for a user who wishes to use a service. For the sake of simplicity, we consider in this chapter the situation where the retailer supplies also the usage (the information retrieval in our example) of the service and not only the access.

Multiple users can use a service via the retailer. It means that multiple users can ask the retailer to fetch information resources. By default, the retailer always fetches the resource from its original location. In order to improve access time, it is useful to move the information closer to the retailer. This problem is a well known problem in the Web today where the notion of proxy has been introduced. The functionality of a proxy is twofold:

- Crossing a network firewall;
- maintaining a cache.

In the TINA architecture, we can consider the retailer as a proxy between the user (consumer) and the content provider. If the first functionality mentioned above is well taken into account in the architecture, mainly by the security aspect, this is not the same for the second point. Indeed, the notion of caching is not explicitly specified in TINA today. It means that, by default, if several users want to access to the same information resource through the same retailer, this retailer systematically has to fetch this resource from its remote host. If instead, the retailer can cache this resource in his own environment (according to some agreements with the content provider), the performances will be improved and the network less loaded.

There is some other advantages to have the retailer acting as a proxy between the user and the content provider for information services. Namely

- More “user-friendly” environment (compared to “open Internet”) by the way of customized pages;
- Possibility of providing a better directory service, as we know today;
- Make subscription on the behalf of the user;
- Service access control: to restrict access for children for instance;
- Statistics;
- Billing;
- Routing; and

- Security.

And more generally, to delegate to the retailer the delivery of mediation services that may be out of the responsibility or expertise of the content provider.

Situation Today

There are different caching mechanisms today. If we consider the WWW, we have two kinds of caching:

- On the user side;
- In the proxy (if there is a proxy between the user and the server).

Proxy caching means that we can use a proxy and a unique cache for several users in the same domain. Both the user and the service provider should be able to manage this cache. For instance, the user may wish to not use any cache and the server can specify the validity (expiration time) of its information in the cache. This is already taken into account in HTTP/1.1 (Cache-Control header) where one section is devoted to caching.

In this chapter, we identify and propose a caching mechanism for TINA partially based on the existing solutions in the Web.

A-6.1.1 TINA Information Repository

This chapter presents a solution for information storage in a TINA context. We use the information repository terminology to express either a cache, a proxy or a server. The reason is that we propose a generic information repository which can be used as a cache, proxy and server in the same way. This information repository can be seen as a TINA facility.

A-6.1.1.1 Business model

The TINA business model defined in [3] and briefly mentioned in Section 2 and depicted (for the service architecture part) in Figure A1-6-1, fits for information services.

Nevertheless, the notion of content provider has been introduced. The content provider is a kind of “lightweight” 3pty Service Provider specialized for information resources delivery in this document (and in the context of the information service task). The content provider may or may not be involved in the usage of the service. For instance, the content provider can supply a movie but not the usage of this movie. It means that the content provider will allow a retailer for instance to offer this movie to his customers. Then, the retailer will “rent” this movie to the content provider and manage its usage. The content provider won’t be involved for the usage but, as explained in Figure A1-6-4, he could update the information resource with a new version through a specific interface supplied by the retailer. This is one of the different aspects of an information repository. A retailer will be allowed to store temporarily information resources for both improving performances and managing more long-term information resources belonging to content provider.

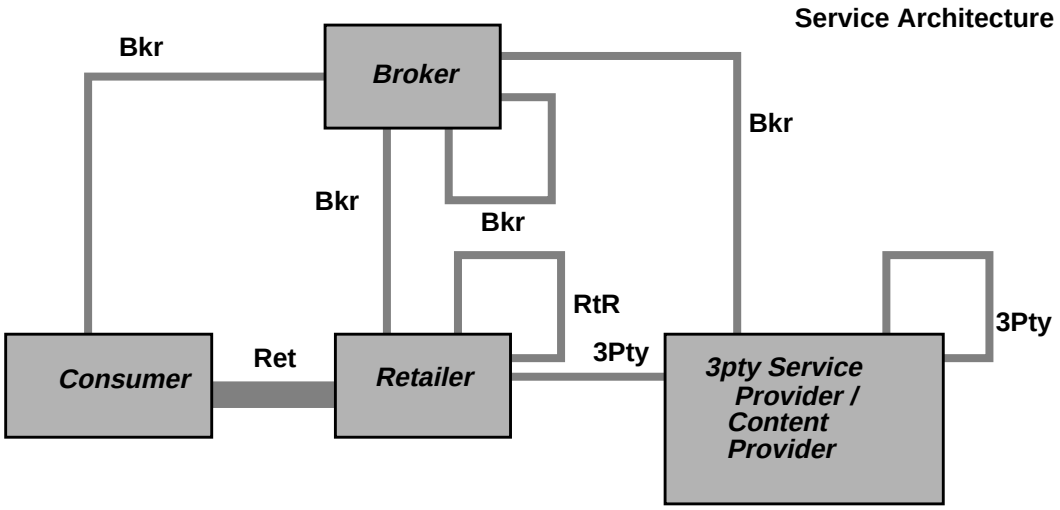


Figure A1-6-1. Business model (service architecture part)

When the content provider is involved in the service usage, the interface offered is, so far, “service dependant”. This interface doesn’t fit for video files where, for instance, a DSM-CC like interface could fit better. So, now if there is a real business behind such services, this kind of interfaces could be prescribed. It wouldn’t be service specific anymore but (family-of-)service generic. Then, we would have different sets of prescriptive interfaces depending of the kind of services.

Either the retailer or the content provider can initiate the access. It means that in some circumstances (the retailer needs an information resource), the retailer acts in a user role and the content provider in a provider role and, in some other circumstances (the content provider wants to update an information resource in the retailer domain), the content provider acts in a user role and the retailer as in a provider role.

All business roles can use an information repository. The Table A-6-1 below shows an example of the usage of information repository by different business roles. This is a simple example which should be reconsidered (evolution, not modification) if we want to take into account service composition and service federation.

Table A1-6-1. Relationships between Information Repository and business roles

		Consumer	Retailer	3PSP / Cont.Prov.	Broker
Information Repository	Cache	X	X		X
	Proxy		X		
	Server		X	X	

A-6.1.1.2 Information Model

Figure A1-6-2 and Figure A1-6-3 depicts the information repository. An information repository can either be a cache, a proxy or a server. What is important, is that they supply exactly the same methods and so, allows a total transparency to the client.

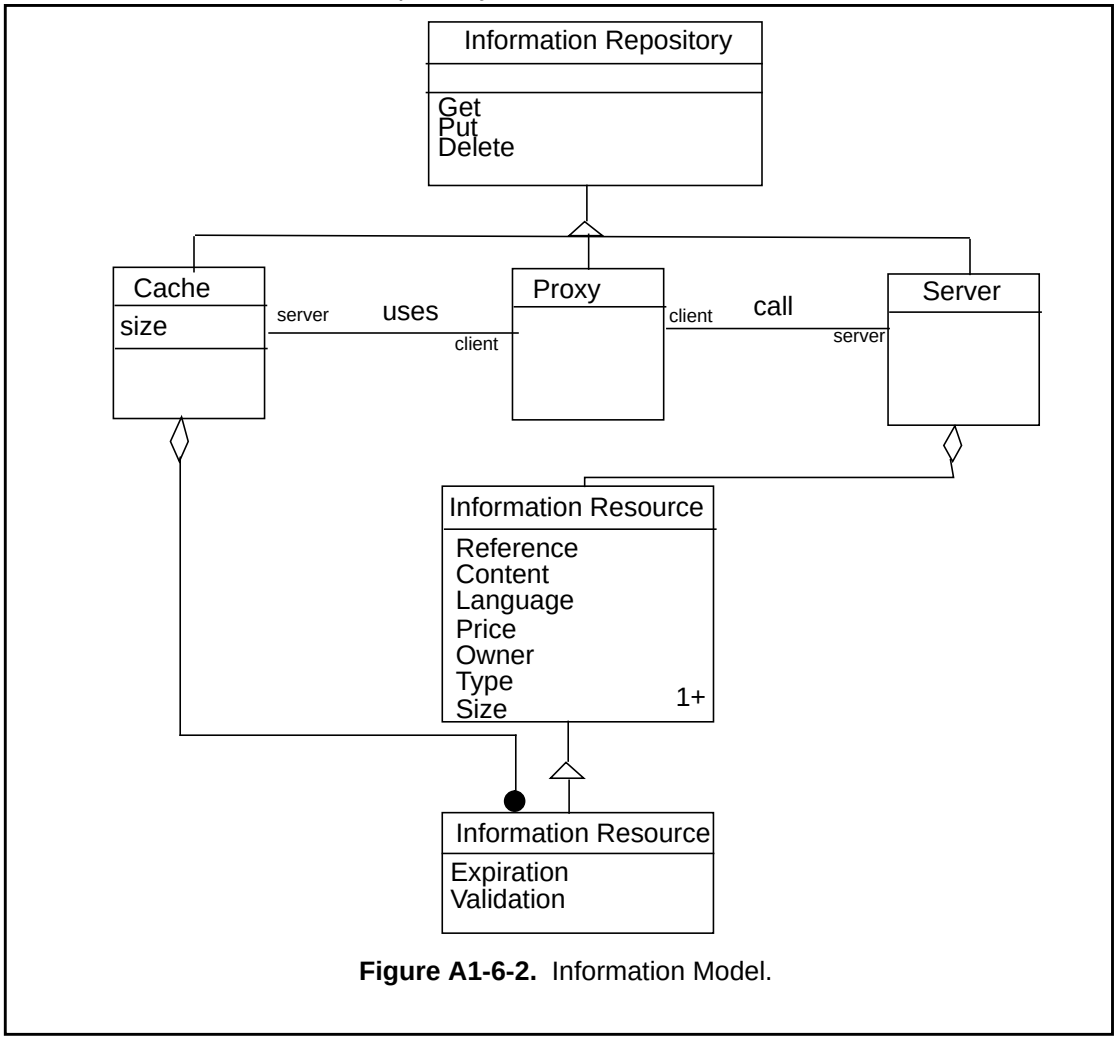


Figure A1-6-2. Information Model.

Methods

Three methods, based on HTTP, define the behavior of an information repository:

- GET: request to fetch an information resource from the cache/server;
- PUT: request to add an information resource in the cache;
- DELETE: request to delete an information resource from the cache.

Of course, an information repository can be specialized further, and then more methods can be defined.

Attributes

The main differences between a cache and a server is the information they contain.

For each information resource it contains, a server supplies the following attributes:

- Reference: the reference of the information resource. Unique in its domain.
- Content: the information resource itself (html file, gif file, etc.).
- Type: the type (format) of the information resource. The client of the cache is not able to interpret the information resource except for type="HTML". Indeed, in order to improve performances, the client of the original server can parse an HTML page to fetch directly all the embedded links. The reader is referred to [28]for more information. The information resource is sent to the user application, which is the only one able to interpret this information based on the format.
- Language: the language of the text if the resource is a text file. This is more an example of the attributes that can be assigned to an information resource. We can also define a generic attribute type which values are closely related to the format. The specialization of the information resource to specific formats has not been deemed necessary and that is the reason why the format attribute is present.
- Price: the price of the information resource. no assumptions are made here on the accounting methods (by act, content, size,...).
- Currency: To be able to interpret the price above.
- Owner: the owner of the information resource.
- Size: the size of the information resource.

A cache supplies some extra information that are useless in a server, namely:

- Reference: this is a contextual name. A name only understandable by the information repository authority.
- Expiration: The expiration time of the information resource. If this time has not expired, the client doesn't have to contact another information repository (proxy or server). It is similar to the expiration model defined in [44].
- Validation: The validation policy used to know if the stale information is still valid. It leads to the establishment of a connection to another information repository, but it doesn't always lead to fetching the information again (depends on the policy). It is similar to the validation model defined in [44].

Another attribute, specific to the cache and not correlated with a specific information resource has to be identified. It is the size of the cache itself.

A-6.1.1.3 Management of an Information Resource in an Information Repository

A1-6.1.1.3.1. Agreements between user, retailer and content provider

Both the users and the content provider can express some requirements concerning the information resources.

The user should be able not to use a cache at all. If he wants to use a cache, he should be able to express some requirements on the freshness of the information resources. For example, the user should be able to say "I don't want any information older than two days". For the sake of simplicity and performance, the requirements are based on all information resources and not on specific ones. These requirements are expressed during the ancillary usage between the consumer and the retailer and stored in the user's profile. During the primary usage, these requirements are added to the initial request of the user in the USM.

The content provider should be able to either refuse to use a cache (for copyright purpose for example) or to use a cache but following some specific policies. These requirements are defined during the ancillary usage between the content provider and the retailer. These requirements can be based on a specific information resource or on a family (a group) of information resources. The latter could be based on all the resources coming from a specific host. By default, an information resource is cachable, including the results dynamically generated (CGI scripts for instance). This ancillary usage between the retailer and the content provider is not mandatory. It means that, if a content provider doesn't want to cache an information resource, he has two ways to express this. Either during the ancillary usage, either by adding a specific tag "no-cache" in the header of the information resource. The latter alternative is only used when the information resource has not been registered in the retailer domain.

As explained in Section A-6.1.1.1, the content provider may allow the retailer to store and deliver the information itself. In this case, the procedure is the same that the one used for caching, with the only difference on the time-out on the information resource which could be "infinite" in this situation.

A1-6.1.1.3.2. Freshness of an Information Resource

This section explains how to consider if an information resource stored in the cache is still valid. The main difference between this solution and the solution adopted by HTTP, is that the freshness of an information resource is calculated on an estimated expiration time and not the current age of the information resource. The reason is that we want to minimize the control mechanisms deemed to heavy in HTTP and consequently, to reduce the size of the headers also deemed to long in HTTP. The expiration and validation models defined below are not HTTP compliant and the semantic is less powerful than the one proposed by HTPP/1.1.

The expiration model is used as follow. When the user makes a request, the value of "*max_date_value*" of the information resource stored in the cache is compared with "*now*". If *max_date_value* > *now*, it means that the information resource stored in the cache is still valid and there is no connection to the remote server. The value of "*max_date_value"* can be "*infinite*". In this case, the information will be always considered as valid till the content provider update it (a new version) using the operation put of the i_RestrictedAccessIR.

The validation model is used as follow. If *max_date_value* < *now*, it means that the information resource maybe not is valid any more. Then, first of all, we check if the user agrees to have a stale information resource. It's known by the attributes "max_stale" of the t_CacheControl structure (previously fetch from the user's profile). If *max_stale* > (*now* - *max_date_value*), then the information resource stored in the cache is still considered as valid. Else, it leads to a control operation from the retailer to the content provider (for the sake of simplicity we don't consider retailer federation) to know if this information is still valid. Indeed, this information was deemed stalled on the "max_date_value" but it is possible that in the remote server this information resource is still valid. If it is, the information resource is not returned back the retailer but the information stored in the cache is used instead; else, a new information resource is returned back to the retailer's cache.

A1-6.1.1.3.3. Size of the Cache

No assumptions are made concerning the way that expired documents are removed from the cache in order to respect the size of the cache. Some solutions, algorithms, already exist today and this choice is left to the designer. One method has been defined, namely "garbage" but this task can also be done implicitly. Nevertheless, it is important to note that some kind of resources can not be remove from the cache implicitly. Indeed, the cache contains mainly two kinds of information resources. The ones stored for performances purpose and the ones stored on the behalf of the content provider (with an infinite "*max_date_value"*). If the former can be removed implicitly, it's not the same for the latter

where an explicit removal has to occur, based on the common agreement between the retailer and the content provider. See Section A-6.1.1.1 for more information about the relationships between the retailer and the content provider.

A-6.1.1.4 Behavior

The behavior of this information repository can be specialized. It means that through the same interface, an information repository can fetch the information either from a database (using a query language like SQL2, SQL3 or OQL) or from a file system. For example, the hints defined in the previous section could be translated in a “select *reference* where *hints*” if the persistent support is a database accepting SQL queries.

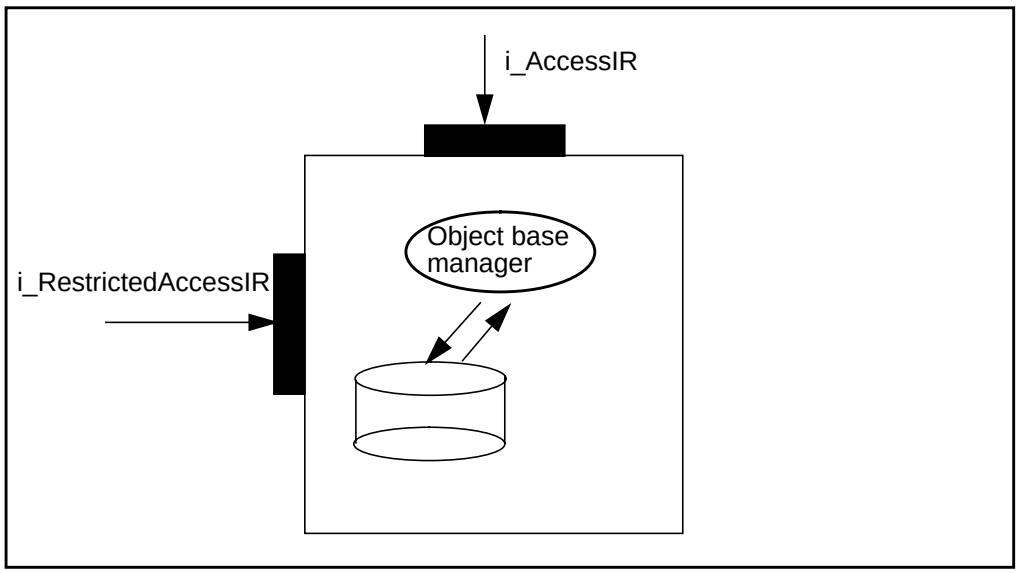


Figure A1-6-3. Information Repository.

No assumptions are made concerning the way that the information is fetched by the Object Base Manager:

- It can be done by using a proprietary query language or using an object query service depending on the DPE support of the information repository.
- It can also be done by using the actual WWW. Indeed, the “reference” (See Figure A1-6-3) can be a URL and in this case, the information repository acts as a gateway to legacy interfaces (HTTP, LDAP, etc.).
- It can also be done by using another information repository (federation of information repositories)

The Object Base Manager translates the operations defined in the previous chapter into a query language (SQL, OQL,...) or into a lower level file systems call.

The reader is referred to [29] for more information on data management.

Dynamic Aspects

Figure A1-6-4 depicts an example of information repository interactions. What is important to note, is that from the client's point of view, there is no difference between a cache, a proxy (the retailer) and a server (content provider). All of them support the same interfaces. The steps are as follows:

1. The consumer requests an information giving a logical name (get() operation). This is a service specific request.
2. The Service Session Manager (SSM) might add some hints to the request based on the user's profile (language, price, cache control, etc.). After updating the request, the SSM looks up this information in its local cache (get() operation). We suppose that this information is not in the local cache or that it is not valid anymore.
3. The SSM asks the Broker Service Manager (BSM) to resolve the reference to a URN resolver providing the resolution service: URN-to-resource - server or proxy. The BSM returns a new reference and the address of a URN resolver (the content provider) that can provide the requested resolution service.
4. The retailer contacts the content provider[1] with an updated request (the reference of the information resource). The content provider returns the information resource to the retailer.
5. The SSM stores (if allowed) the information resource (put () operation) in its local cache and assigns it a date. The information resource might be charged for at this stage.
6. The retailer returns the information resource to the user either in a synchronous way (result of the invocation) or in an asynchronous way.
7. This step is depicted to show an example of the usage of the i_RestrictedAccessIR. This interface can be used by a content provider who wants to "*put*" a new version of an information resource on the retailer domain. The consumer shouldn't have access to this interface.

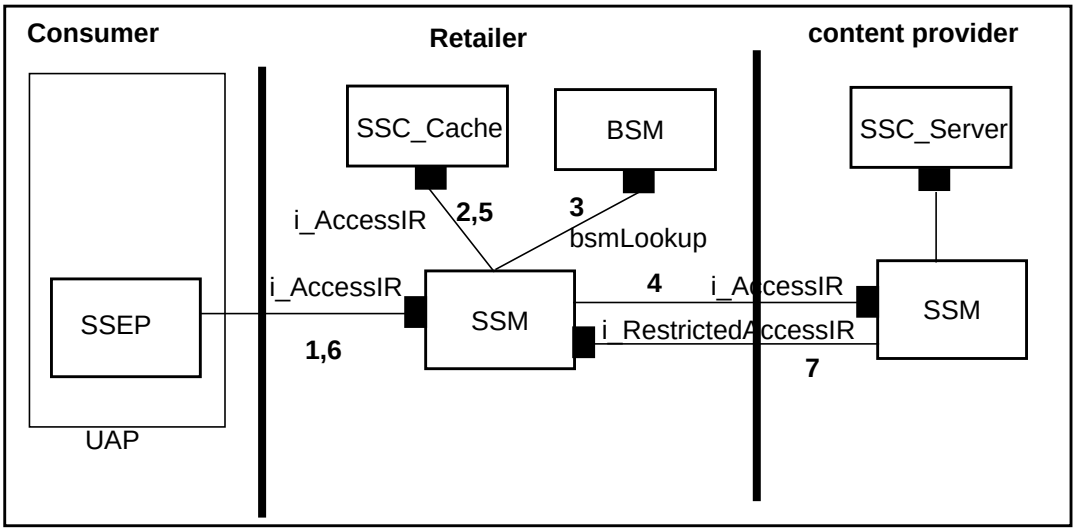


Figure A1-6-4. Example of Information Repository Interactions.

1. The interaction between the retailer and the content provider is handled by the SSM on the content provider side. According to composition and federation work it might also pass through a MUSM (i.e through one one the concrete subslasses thereof).

An information repository might be an information resource server and could typically be a service supplied by a content provider. This also means that the Access_IR interface could be a reference point for the usage of a service between a retailer (or a consumer) and a content provider.

We have mainly investigated the usage of a cache on the server side (retailer in our example). Like in the Web today, we can use this cache on the consumer side as well (as depicted in Table A-6-1). In this case, the call to the cache can be done from the SSEP (if we consider this operation as service specific). We could for example use the “smart proxy” supplied by Orbix (IONA’s DPE) to call the cache. This mechanism allows the designer of the application to take advantage of the server’s proxy on the client side to intercept a request and trigger for example a call to a cache to avoid eventually a remote connection.

A-6.1.2 Integration with Legacy Applications

The methods defined in the previous chapter are based on HTTP methods and are used in a Corba context, built, for example, on top of IIOP. The advantage is that the client doesn’t know where the information comes from and allows the information repository to be totally open for all kind of information resource support (database, WWW server -> http, ftp, gopher, wais, etc.) by just specializing its behavior.

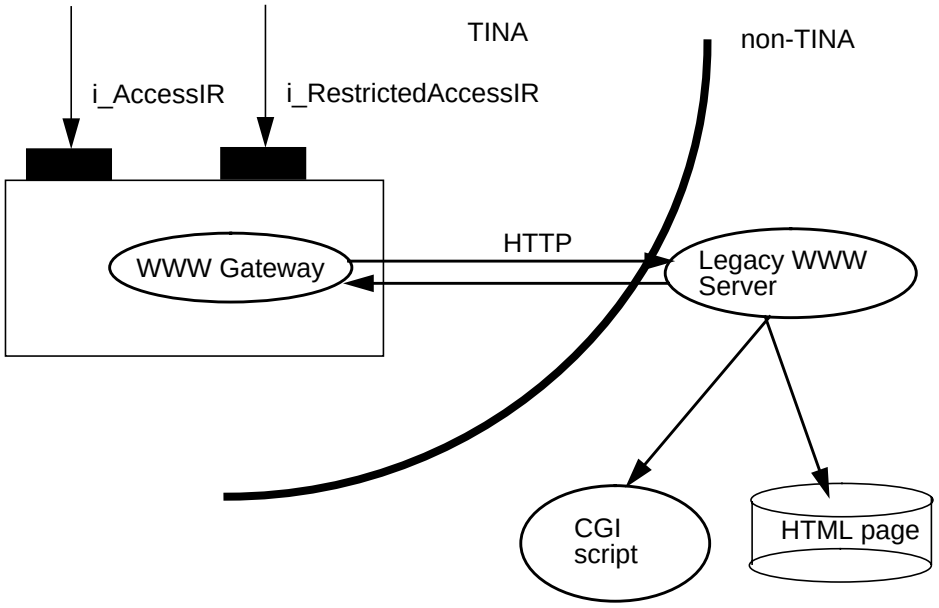


Figure A1-6-5. Integration with Legacy Applications.

The figure above depicts an example of how we could specialize an information repository to fetch a resource from the web. The interface of the information repository is still an ODL interface but the behavior in this case is different because the information repository doesn’t look up this information from a local repository (as depicted in Figure A1-6-3) but from an external legacy repository. The figure depicts an example with the WWW, but any other legacy interface could be used (X.500/LDAP, CMIS, etc.).

If a WWW gateway is used, some HTTP headers (such as the Cache-Control general header field, the Content_Type, etc.) are considered for updating the information repository to be able to manage the cache as defined in Section A-6.1.1.3.

A-6.1.3 Conclusion

As explained in the previous section, the solution presented in this chapter can seamlessly take into account the existing resources on the WWW via HTTP. However, the solution is heavily based on the (quasi) separation between the control and the sending (retrieval) of the information resource, and it is one of the major difference comparing to the Web today where HTTP has to carry very big headers on each requests/response. It's possible because the different entities involved in the information resource delivery are known. This is not the case in the Web. This leads to a better control of the resource than in HTTP and only some minimal headers have to be identified for information sending. For instance, a retailer has to be able to deliver the information resource to several users. It means that when the retailer requests an information resource from a content provider, a request id has to be added in the header of the information flow to be able to give this information resource the right users. However, this header has to be as little as possible to avoid the large size headers of HTTP.

A-7 Other

A-7.1 Service Classification

The classification given here is *one* possible way of classifying services (or 'service features'), it is influence by work from ITU. There may be other classifications as well. E.g. in TINA descrete mobility is handled via UA in the access phase, and the service for registering user at a terminal may as well be classified as an information service. This also indicate that information services may deal with everything from video content to handling of mail messages and even smaller data (like the ones handled by a HLR (Home Location Registar) in GSM).

The service classification presented in this chapter aims at covering all stakeholder services in TINA. The service classification is done from the user's point of view. The service classification shall be used by different TINA stakeholders to advertise and retrieve service offerings via TINA brokers/ traders. The same service name can be used in requests to different TINA business roles and, therefore, be interpreted in different contexts.

This service classification is rather a service feature classification. A service feature is a specific aspect of a service that can also be used in conjunction with other services/service features as part of a commercial offering. It is either a core part of a service or an optional part offered as an enhancement to a service. The TINA service session model allows several service specific information objects to interact with the generic service session model. A TINA service is the combination of these services features.

In the main body of the service architecture, a separation in ancillary usage and primary usage service sessions are made based on the independent value and contractual purpose of the service. Both ancillary usage and primary usage services sessions can be categorized in the service classification below and, as is the case for service features, they can together form a complete commercial TINA service.

TINA services are classified in three main categories:

- Telecommunication services,
- Management services,
- Information services.

These services can be bundled in such a way that information services can use telecommunication services, and management services can be added to telecommunication services and information services to finally constitute only one service from the user's point of view. For example, an information retrieval service can use information services to access to the resource (information content services), and use a telecommunication service to transport this resource. Management services such as for example accounting, can use telecommunication services to charge for the transport and information services to charge based on the content (the resource itself). It means of course that this service is supplied by different stakeholders from different business roles.

A-7.1.1 Telecommunication Services

Telecommunication services are divided in:

- **Bearer services:** provide means to convey information between telecommunication network users without alteration of the communication path or function;

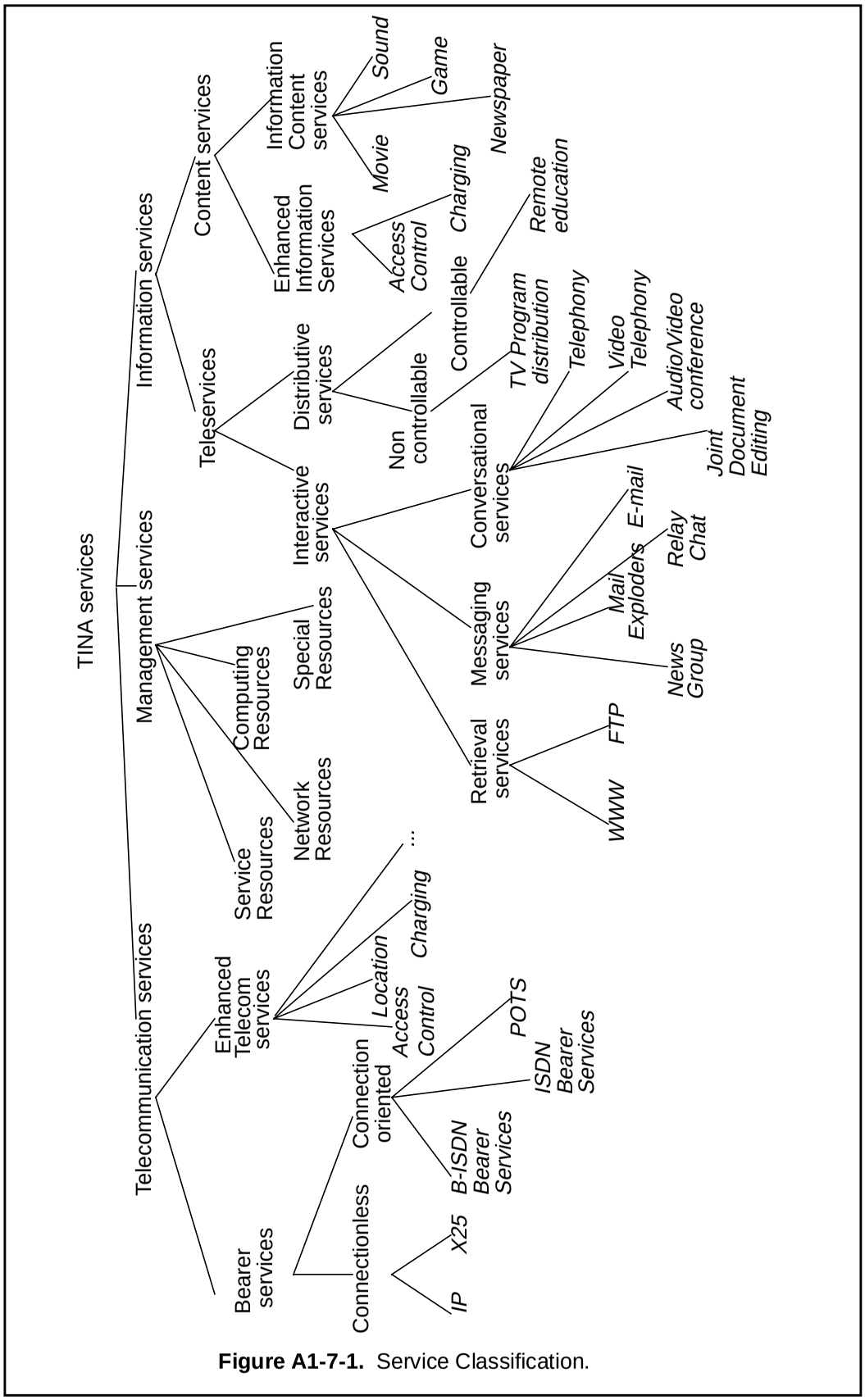


Figure A1-7-1. Service Classification.

- **Enhanced Telecommunication services**: provide additional features by modifying parameters or functions in the basic communication establishment between network users. Intelligent network services (freephone, upt,...) and ISDN supplementary services are typical examples of the latter service category. Mobility services are also considered to be in this category.

Bearer services are divided in:

- **connection-oriented services**: POTS, B-ISDN,...
- **connectionless services**: IP or X25.

A-7.1.2 Management Services

Management services are decomposed into four areas based on the resources they manage: **service**, **network**, **computing** or **special resources**. Example of **special resources** are video bridges and information resources. The TMN management functions (FCAPS) are used to decompose each of these areas further.

A-7.1.3 Information Services

The **information services** is divided into:

- **Teleservices**: represent applications deployed in the terminals that are able to interpret the transfer format.
- **Content services**: include the actual information content. A service provider can, e.g., supply the teleservice, while the content service is supplied by a different stakeholder. Depending on the information content, different subscription, accounting or access restriction services can apply. These services are defined as **enhanced information services**.

The **teleservices** are specialized in:

- **Interactive services:** categorized in
  - **Retrieval services**: provide communication between a user and an information center (server).
  - **Messaging services:** provide communication between at least two users via storage devices.
  - **Conversational services**: provide real time, bidirectional communication between at least two users.
- **Distributive services** are broadcast services with user control (e.g. remote education and training) or without user control (e.g. TV program distribution).

Content services represent services based on the content of the information. They are divided in:

- **Enhanced Information services:** mediation services based on the content of the information, e.g. access control based on the content (the type of movie or document), accounting (per act independent of the content, per act dependant of the content, per volume, etc.). These services are analogous to enhanced telecommunication services, which are independent of the information content
- **Information content services**: e.g., game, movie, sound and document.

Attributes will be attached to the services to specialize them further. Different service providers can supply the same service but with different QoS attributes. Two Information content service offerings can have the same movie but in different languages. In this case, the attribute is the language and is used when trading for a suitable service. Bearer services can have attributes for transfer rate (64kb/s, 2x64kb/s, 384kb/s,etc.), symmetry, encoding, channel connection mode, timing, and information type.

The TINA business roles will be used to give a context in which the service names shall be interpreted. Most service classes can be mapped to a unique business role. Connection-oriented transport services are e.g. supplied by connectivity providers, location services by brokers, enhanced telecommunication services by retailers and information content by service providers.

Figure A1-7-1 shows an inheritance tree relating the TINA service classes. A final user application is often a composition of TINA services, e.g. bearer service + teleservice + content service, involving different stakeholders with different business roles. The leafs in italics give some examples of services but are not exhaustive. The classification can be specialized further. For example, the information content service class “newspapers” can be specialized in sports, politics, economy, etc.

A-7.2 The Universal Service Component Model

This section describes a structure and guidelines that can be useful in order to design, analyze, use, and manage services.

The following *aspects* or views can be used to describe **any** service.

- **Core**: Every service must be identified as to its primary value to a user. This value or application describes the nature of the service regardless of the characteristics of how it is used, how it is managed, or the technology upon which it depends.
- **Usage**: User interfaces[1] to the service must be identified in terms of their requirements upon the service, their behavior, and appearance to the core and to external users of the service.
- **Management**: The requirements for and techniques of providing management (operation, provisioning, administration, and maintenance) must be defined.
- **Substance**: The interaction and dependence of the service upon external resources and other services must be defined.

1. The term user interface is used here in a broad sense, and includes, but is not restricted to, Human Computer Interfaces and Application Programming Interfaces.

Each TINA-C service should take into account the aspect separation compliant to the Universal Service Component Model, depicted in Figure A1-7-2.That is Access, Core, Management and Substance aspects are to be defined separately.

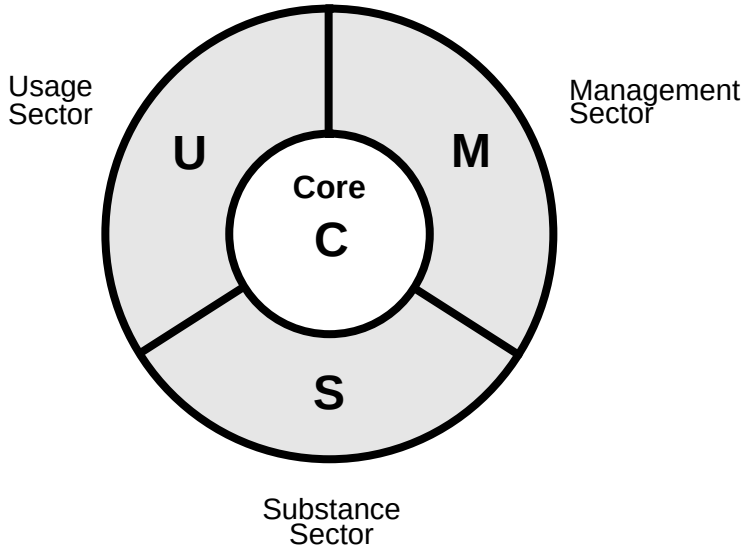


Figure A1-7-2. Structure of the Universal Service Component Model.

These aspects represent more than just perspectives upon a service; they can also be used to categorize *service components* and interactions between services. A service component is a self-contained unit of service construction, and provides an identifiable part of the service. The functionalities and required information of one service component can be identified as separate from the functionality and information of other service components in the same service. Service components offer accessible interfaces. This concept allows services to be specified, defined, designed, built, and managed in a modular fashion. Such a modularity will simplify creation, deployment, and management of services while reducing interaction problems caused by side effects and data sharing.

The primary goal of each component is to focus on one aspect of services like Core, Access, Management or Resource. The following guideline can be stated: each TINA service component provides information and functionalities that do not overlap with those offered by other service components.

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 is consistent with the USCM division and is therefore the same as other TINA compliant services.

The USCM is an abstraction of all TINA services. It provides both an internal and external description of the structure of any TINA service. The relationships between USCM components describe constraints on the group of related components that comprise the service. This common service format is specified to promote consistency, reuse, and simplification of management.