INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Series Q
Supplement 29
(12/1999)

SERIES Q: SWITCHING AND SIGNALLING

Service modelling: Evolution to the use of object oriented techniques

ITU-T Q-series Recommendations – Supplement 29

(Formerly CCITT Recommendations)

ITU-T Q-SERIES RECOMMENDATIONS

SWITCHING AND SIGNALLING

SIGNALLING IN THE INTERNATIONAL MANUAL SERVICE	Q.1–Q.3
INTERNATIONAL AUTOMATIC AND SEMI-AUTOMATIC WORKING	Q.4–Q.59
FUNCTIONS AND INFORMATION FLOWS FOR SERVICES IN THE ISDN	Q.60–Q.99
CLAUSES APPLICABLE TO ITU-T STANDARD SYSTEMS	Q.100–Q.119
SPECIFICATIONS OF SIGNALLING SYSTEMS No. 4 AND No. 5	Q.120–Q.249
SPECIFICATIONS OF SIGNALLING SYSTEM No. 6	Q.250–Q.309
SPECIFICATIONS OF SIGNALLING SYSTEM R1	Q.310–Q.399
SPECIFICATIONS OF SIGNALLING SYSTEM R2	Q.400–Q.499
DIGITAL EXCHANGES	Q.500–Q.599
INTERWORKING OF SIGNALLING SYSTEMS	Q.600–Q.699
SPECIFICATIONS OF SIGNALLING SYSTEM No. 7	Q.700–Q.849
DIGITAL SUBSCRIBER SIGNALLING SYSTEM No. 1	Q.850–Q.999
PUBLIC LAND MOBILE NETWORK	Q.1000–Q.1099
INTERWORKING WITH SATELLITE MOBILE SYSTEMS	Q.1100–Q.1199
INTELLIGENT NETWORK	Q.1200–Q.1699
SIGNALLING REQUIREMENTS AND PROTOCOLS FOR IMT-2000	Q.1700–Q.1799
BROADBAND ISDN	Q.2000–Q.2999

For further details, please refer to the list of ITU-T Recommendations.

Supplement 29 to ITU-T Q-series Recommendations

Service modelling: Evolution to the use of object oriented techniques

Summary

This Supplement compares different types of methodologies for service modelling to determine their suitability in relation to protocol development for IN in the IN CS-4 time-frame. It also investigates the evolution from SIB based techniques and considers different technologies such as APIs.

This Supplement supplements the information contained within Recommendation Q.65.

Rational

The purpose of this Supplement is to provide a discussion of service modelling aspects related to IN CS-4.

Source

Supplement 29 to ITU-T Q-series Recommendations was prepared by ITU-T Study Group 11 (1997-2000) and approved under the WTSC Resolution 5 procedure on 3 December 1999.

CONTENTS

Supplement 29 to ITU-T Q-series Recommendations

Service modelling: Evolution to the use of object oriented techniques

1 Scope of service modelling for IN CS-4

The purpose of this Supplement is:

- to identify and compare different methodologies for service modelling;
- to determine the suitability of the identified methodologies for service modelling, in particular, with respect to:
  - service management: data modelling;
  - service logic purposes: dynamic service model (behaviour of service logic);
  - description of the relationship between the data model and the dynamic service model.
- to determine the suitability of the identified methodologies for protocol development, in particular, with respect to:
  - the support of stepwise refinement from service and network capabilities to protocol level both for the data model and the dynamic service model;
  - the ability to incorporate the existing protocol into the model defined with the used methodology.
- to investigate the migration aspects of service modelling, in particular, with respect to:
  - the evolution from the SIB-based service modelling methodology to the methodologies used for IN CS-4.
- to decide on the scope of service modelling for IN CS-4;
- to decide on the methodology to be applied for service modelling in IN CS-4.

1.1 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Supplement. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; all users of this Supplement are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations and supplements is regularly published.

- ITU-T Recommendations Q.122x series, *IN CS-2 Recommendations.*
- ITU-T Recommendation Q.1223 (1997), *Global functional plane for intelligent network capability set 2.*
- ITU-T Recommendation Z.100 (1999), *Specification and description language (SDL).*

2 Definitions and abbreviations

2.1 Abbreviations

This Supplement uses the following abbreviations:

API	Application Programming Interface
GFP	Global Functional Plane
INCM	IN Conceptual Model
ODP	Open Distributed Processing
OMT	Object Modelling Technique
SDL	Specification Description Language
SQL	Structured Query Language
STD	State Transition Diagrams
TINA	Telecommunication Information Networking Architecture

2.2 Definitions

This Supplement defines the following terms:

2.2.1 API: an API is essentially a set of operations (or methods) that can be invoked on a component, each of which causes the component to exhibit behavioural functionality. Each operation is specified syntactically as an identifier which identifies the operation being invoked, and parameters which affect the behaviour of the component in some way.

2.2.2 API Call: an API call is equivalent with the term "operation" and is sometimes interchangeably used in the description of an API (above). Effectively an API is made up of a number of individual API calls.

3 Requirements for IN CS-4 service modelling

This clause looks at the requirements that are seen to be important for modelling techniques and methodologies used for IN service modelling, they are:

- **Stepwise refinement** from service (features) to protocol level should be supported.
- Support for various mechanisms of **transparency.** These include:
  - *Technology* transparency: the modelling of IN services should not be dependent on the technology used, e.g. the type of network, operation system or programming language. An important type of transparency in the context of IN CS-4 service modelling is network technology transparency. The objective of IN CS-4 is the integration of a number of network technologies, including connection oriented type of networks (e.g. PSTN, IMT-2000) and connectionless types of networks (e.g. IP-based networks, data networks). Some of the services targeted in IN CS-4 will probably be specific to one network technology, others will require interworking between different network technologies, or their behaviour will depend on the technology the service is invoked from. One could think of a service with user interaction, which can be invoked by B-ISDN as well as mobile network users, or of a multimedia conference service over different network technologies. IN could be an integrated architecture for the services on multiple network technologies. It should be investigated to what extent network technology transparency can be achieved.

  - *Access* transparency: masking differences in data representation and invocation mechanisms (i.e. providing multiple mappings of the information contents of IN protocol exchanges to programming APIs).
  - *Failure* transparency: masking, from an object, the failure and possible recovery of objects (including itself).
  - *Migration* transparency: masking, from an object, the ability of a system to change the location of interfaces to that object.
  - *Relocation* transparency: masking relocation of an object interface from other interfaces bound to it.
  - *Replication* transparency: masking use of a group of mutually behaviourally compatible objects to support an interface.
- **Service extensibility**: it should be possible to support additions to the IN service functionality by extending the existing service models and existing service components.
- **Service scalability**: it should be possible to accommodate and allow to scale the service capabilities in terms of the number of users, the number of nodes, the number of administrative domains, etc.
- **Service version handling**: service-modelling techniques should facilitate the concurrent use of multiple versions of service models and service components.
- **Service reusability**: it should be possible to reuse the models of an IN service (capability) during the specification of another IN service, rather than starting the modelling from scratch even in case of overlapping functionality. The same applies to the software components that implement the specification.
- **Operation, administration and maintenance**: the methodologies and modelling techniques should support flexibility when it comes to functionality and data model changes during operation, administration and maintenance. For example, functionality changes that occur during maintenance should be easily reflected in the model of the IN service.
- **Reduction of service conflict**: the methodologies and modelling techniques should reduce the risks of service interactions.
- **Flexible service modification/customization**: the network providers and service providers should to be able to differentiate and customize services to meet specific market needs.

  The Users should be able to customize the services to some extent to meet their personal needs. In the short term it will be limited to data customization and selection from a list of predefined features. In the longer term service packaging might be provided as well.
- **Support for a multi-stakeholder environment**: the IN CS-4 service capability modelling should be able to take into account the respective roles of the different stakeholders involved in the service provisioning (retailer, operator, content provider, etc.).

3.1 Service modelling

One of the goals of OO modelling is to allow the service developer and designers to share the same model of Service Logic. In order to achieve this goal the modelling technique should hide all platform details from the user but at the same time allow the service developer to translate the behaviour model into the suitable execution form. In order to meet those requirements it is proposed that the behaviour modelling technique be represented in the notation which is programming language, operating system and database independent. The use of data-less SDLs and State Transition Diagrams is thus proposed to represent pure behaviour of an abstract class.

3.2 Service Logic spanning a single class

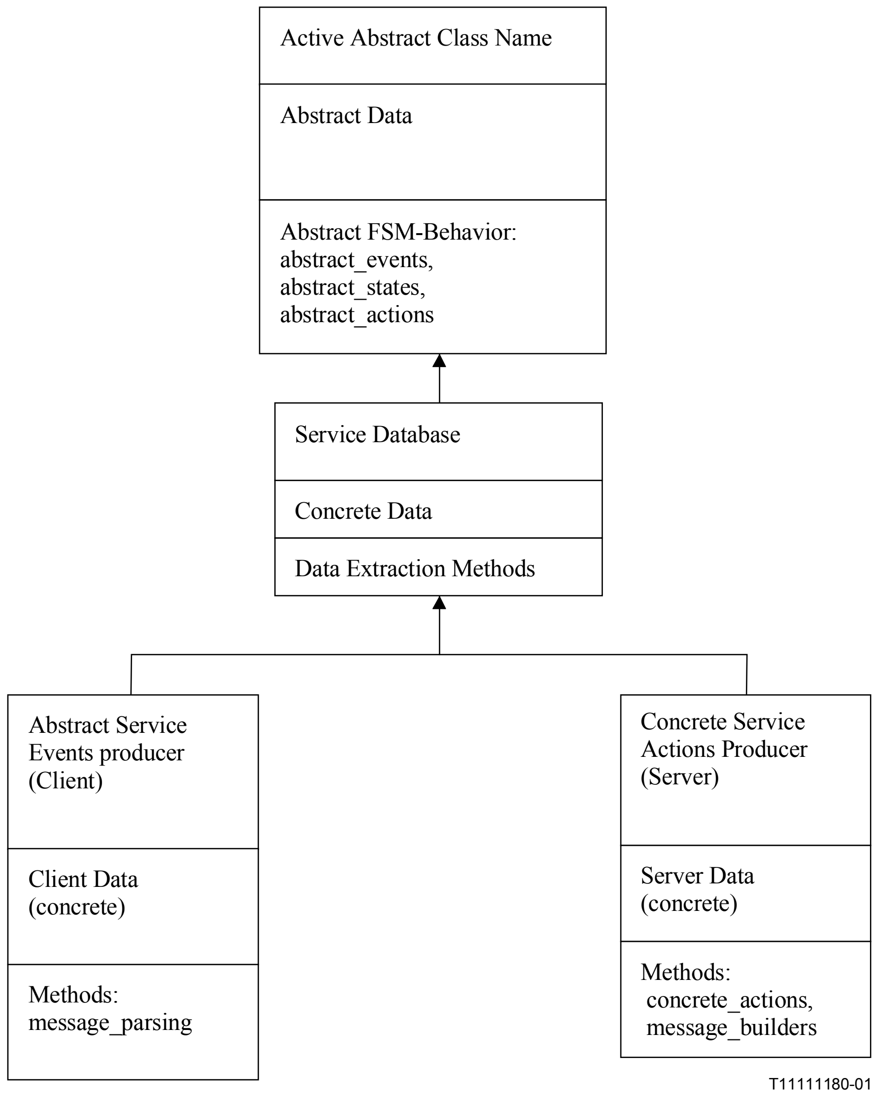
The OMT and other modelling techniques introduce the concept of an Active Class. An Active Class is defined as a Class whose behaviour is modelled best using the Finite State Machine Model (FSM). In order to achieve the language independence at the service logic level it is proposed that the model of Finite State Machine used contains no data. Thus an Abstract Class representing a simple service would contain an Abstract Data and an Abstract Behaviour Model strictly separated.

It is important to stress here the fact that the separation of an Abstract Data and an Abstract Behaviour Model is suggested only at the Modelling Level. No recommendation is being made here nor is it suggested that the separation of an Abstract Data and an Abstract Behaviour Model shall be extended to the automatically or manually generated code.

An example of a separation of an Abstract Data and an Abstract Behaviour could be a VPN service with a complex service model. For instance the VPN service parameter Participant_ID may be an example of abstract data at the modelling level. The Abstract Class Model in its Abstract Data Section should not reflect where in the behaviour model this data is used. On the other hand, the Abstract Behaviour Model of an Abstract Class may require an event New_Participant_Arrived without the need to mention where data for a given participant is stored in the Abstract Data Section of the Abstract Class.

This separation within a top-level class should allow for inter-changeability of Service Model Objects and a creation of standard service needs. See Figure 1. The arrows in this model represent inheritance in a manner consistent with the OMG notation. In the case of C++ implementation abstract_actions and abstract_events are often implemented as pure virtual functions.

Figure 2 shows an example of an Abstract Active Class called VPN Service Object, which may be used for the implementation of Virtual Private Network service. The class contains examples of abstract_events, abstract_actions and abstract_states.


Active Abstract Class Name
Abstract Data
Abstract FSM-Behavior: abstract_events, abstract_states, abstract_actions
Service Database
Concrete Data
Data Extraction Methods
Abstract Service Events producer (Client)
Client Data (concrete)
Methods: message_parsing
Concrete Service Actions Producer (Server)
Server Data (concrete)
Methods: concrete_actions, message_builders
T11111180-01

NOTE – It is feasible that the message_parser and the message_builder may be implemented as separate classes for more complex services.

Figure 1 – An abstract class representing: a) a simple Service Logic; b) a portion of a complex Service Logic; c) a Service Logic Manager

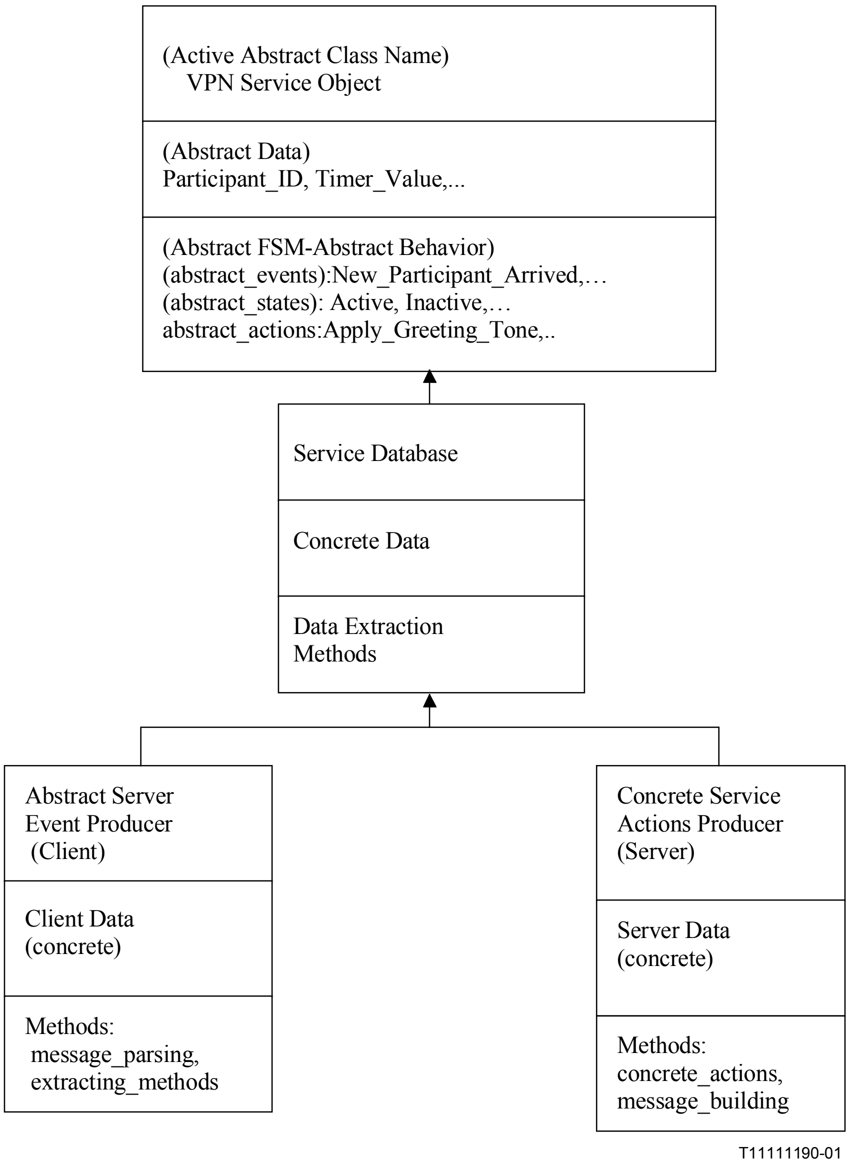

(Active Abstract Class Name)
VPN Service Object
(Abstract Data)
Participant_ID, Timer_Value,...
(Abstract FSM-Abstract Behavior)
(abstract_events):New_Participant_Arrived,…
(abstract_states): Active, Inactive,…
abstract_actions:Apply_Greeting_Tone,..
Service Database
Concrete Data
Data Extraction Methods
Abstract Server Event Producer (Client)
Client Data (concrete)
Methods: message_parsing, extracting_methods
Concrete Service Actions Producer (Server)
Server Data (concrete)
Methods: concrete_actions, message_building
T11111190-01

Figure 2 – An example of abstract class representing a VPN Service Logic

3.2.1 Service Logic spanning several classes

For the more complex services, Service Logic may be modelled as a collection of cooperating Active Classes. An object is defined here as an instantiated class. It may be worthwhile to investigate whether the model of cooperating classes is more appropriate for the IN Service Modelling than the model of contained classes.

There are indications that the model of cooperating classes better enforces the encapsulation of classes hiding the implementation details. Such a model of cooperating classes was a foundation of Smalltalk-Object programming language. In such a case each of the active classes should be modelled in the same way as the case of Service Logic spanning a single class. The cooperating classes send requests to each other and receive responses. This represents a flow of control information between the objects.

It should be noted that a term message, as used in this contribution, implies physical rather than logical flow of information. A message may contain a data as well as a control information. Thus it is recommended here that flow of control information between the objects be shown separately from the flow of data between the objects. A known advantage of such a separation is a simplification of tools validating the service automatically. The Service Creation Environment (SCE) validation tool should generate sequence flow diagrams of inter-objects communication from FSM behaviour models of cooperating objects. Further contributions on SCE and SMS requirements could be provided in a later date. The model presented shall apply to a peer type inter-objects communication as well as to the situation where one of the classes is assigned a function of the Service Logic Manager. This corresponds to the Service Designer choice of using hierarchical or rather decentralized form of Service modelling.

4 Methodologies and modelling techniques

4.1 Open Distributed Processing (ODP)

The reference model of ODP is a generic distributed object-oriented methodology that is suitable for both traditional telecommunications applications (such as IN) and information processing applications. It is based on two powerful trends in software technology:

- Object-oriented specification techniques provided at different abstraction levels allowing a high degree of stepwise refinement and consistency checking.
- Object-based distributed processing environments (e.g. CORBA-based platforms) allowing the provision of distributed services, and most importantly, enabling a distribution transparency and interworking within distributed systems.

The reference model of ODP prescribes that multiple descriptions, with different abstraction levels, should be provided for any service under study or development. Five abstraction levels – called *viewpoints* in the ODP terminology – have been defined within ODP and are considered to encompass the different areas of concerns that need to be covered in the service development process.

Specifications have to be provided in each of these viewpoints using a corresponding viewpoint model or an adequate viewpoint language. The five ODP viewpoints are *enterprise*, *information*, *computational*, *engineering* and *technology*.

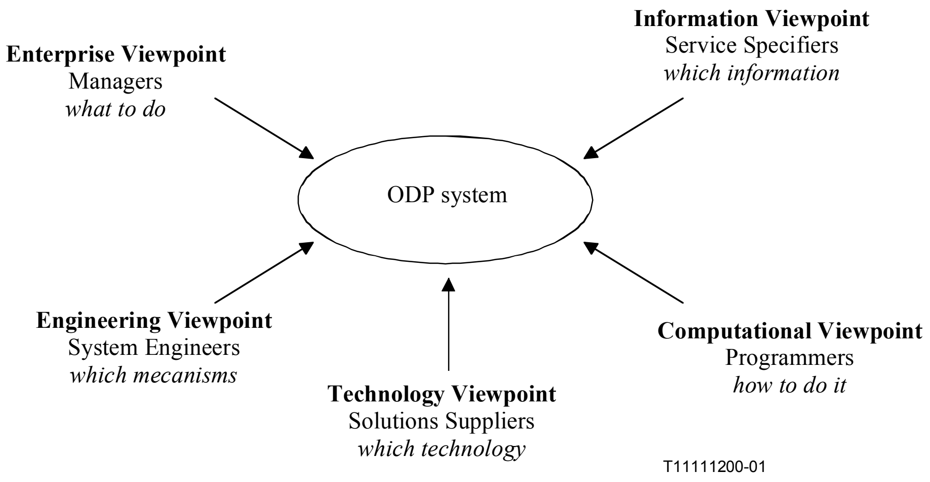


Figure 3 – Different abstraction levels (viewpoints) in ODP

4.1.1 Enterprise Viewpoint

An enterprise specification of an IN service describes the service from the perspective of the organizations and people that will use or operate that service. The concepts used for enterprise specifications include:

- the requirements that state the desired capabilities of the service at a strategic level (why it is being considered);
- the stake-holders (and other participating institutions);
- the roles (who will be involved and in what role);
- the legal environment (obligations of the stakeholders);
- the set of rules and regulations that govern the exploitation of the service (the rules that must be obeyed when the service will operate).

Usually an enterprise specification is written in a textual format or using a high-level graphical notation.

4.1.2 Information Viewpoint

An information specification provides a highly abstract model of real world entities and their relationships along with the constraints that govern their behaviour. An information specification typically covers:

- the specification of the information structure of the service (objects);
- the specification of the dependencies between the information objects (associations);
- the specification of the operations and constraints that govern the possible dynamic evolution of the service considered as collection of objects.

The information viewpoint could be developed using e.g. UML, especially using the capabilities for data modelling (static model).

Guidelines for mapping Enterprise viewpoint to Information viewpoint:

- Mapping rules can be established to translate the enterprise specification into information terms. It should be noted that these are only guidelines.

4.1.3 Computational Viewpoint

A computational specification represents an abstract implementation of the service under consideration in terms of interacting objects whereby location, access, distribution and failure are transparent. Put simply, at this abstraction level, a service is represented as a dynamic configuration of interaction objects.

An important feature of computational specification is the ability to capture the real-time and probabilistic aspects. Binding objects, for instance, describe the behaviour of communication which complies with certain QOS constraints. These non-functional requirements are particularly relevant when it comes to the support of multimedia interactions in a distributed environment.

The specification of the computational viewpoint would typically contain the well-known functional entities such as SCF, SSF, SRF, etc. as computational objects, perhaps further decomposed depending on the distribution needs. The interactions between the computational objects would be in terms of "operations" (i.e. INAP operations) and "flows" (i.e. voice, video and data).

The computational viewpoint could be developed using e.g. UML for specification of the computational objects and e.g. IDL for the specification of interfaces between objects.

Though ODP does not recommend any mapping solution between information and computational object types, the following simple guideline is a starting point: there is a one-to-one mapping. A computational object type specification is obtained by taking an information object type specification and adding the description of the operations for which it can have a server role. However, considerations regarding the potential for distribution will modify this mapping.

4.1.4 Engineering Viewpoint

An engineering specification determines how computational, distribution-free descriptions can be realized in terms of generic system components and communication protocols (such as SS7). It therefore focuses on how interaction between objects is achieved and which resources are needed to do so. From an engineering viewpoint, an ODP system is considered as a collection of computer systems. The details of the underlying communication networks, operating systems and hardware are hidden by a uniform and basic distributed environment.

In ODP terminology the SSF-FSMs and the BCSMs could be considered as *stubs* or *protocol objects*. The signatures of the operations would correspond to the ASN.1 definitions of the INAP operations.

The Engineering specification would be very similar to the protocol specification.

It could be developed using SDL and ASN.1. SDL is used to describe the objects and their behaviour, and ASN.1 to describe the signatures of the operations visible at external interfaces (hence corresponding to protocol messages). The reason for using ASN.1 and not IDL is that a large number of protocols with which IN needs to inter-work are already specified using ASN.1.

The consistency between the Computational and the Engineering specifications can be maintained by using tool support. For example, the computational objects, specified using UML/OMT, can be "pasted" in the SDL specification as SDL objects (types). The tool will maintain such links between UML/OMT and SDL objects, and consistency checks can be done between the two models. Also, SDL process diagrams can be automatically generated from UML/OMT state chart descriptions.

4.1.5 Technology Viewpoint

The technology viewpoint describes the implementation of the system in terms of the hardware and software components. It may need to consider cost and availability constraints. Selections influence the performance and quality of service of the system. Because it is directly concerned with implementation, the technology viewpoint is outside the scope of standardization.

4.2 Evaluation of ODP

The benefits of ODP for IN service modelling include:

- *Stepwise refinement of specifications*: shows some possible scenarios of specification refinement in ODP (see Figure 4). Note that specification refinement and transformations can be made in an iterative manner.

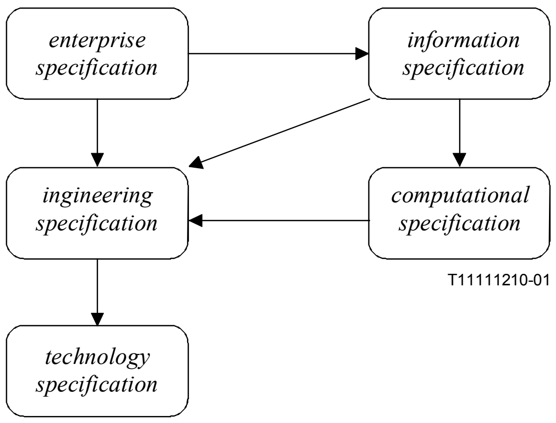

enterprise specification
information specification
ingineering specification
computational specification
technology specification
T11111210-01

Figure 4 – Possible scenarios of specification refinement in ODP

- *Transparency mechanisms*: ODP provides various mechanisms of transparency at different levels of abstraction. These transparency mechanisms include: access transparency, location transparency, failure transparency, migration, transparency and transaction transparency.
- *Service portability* and *Reusability of service components*: these two requirements can be easily supported in an object-oriented DPE, such as a CORBA-based platform.
- *Feature Interaction*: ODP strongly advocates use of Formal Description Techniques (FDT) in different viewpoints. Use of FDTs enhances the description of service behaviour and considerably reduces the risks of service interactions.
- *Naming and Trading Functions*: these are parts of the functions defined by ODP. The trader function provides the means for advertising service offers and the means to discover service offers through service requests.

ODP does not provide any method on how to develop each viewpoint or which language should be used to describe it. Therefore ODP should be applied in conjunction with the appropriate modelling techniques for each of the viewpoints.

4.3 Unified Modelling Language

Introduction to UML

The Unified Modelling Language (UML) is a modelling language that incorporates the object-oriented community's consensus on core modelling concepts. It allows deviations to be expressed in terms of its extension mechanisms. The developers of UML had the following objectives in mind during its development:

- Provide sufficient semantics and notation to address a wide variety of contemporary modelling issues in a direct and economical fashion.
- Provide sufficient semantics to address certain expected future modelling issues, specifically related to component technology, distributed computing, frameworks, and executability.
- Provide extensibility mechanisms so individual projects can extend the metamodel for their application at low cost. Users should not be forced to adjust the UML metamodel itself.
- Provide extensibility mechanisms so that future modelling approaches could be grown on top of the UML.
- Provide sufficient semantics to facilitate model interchange among a variety of tools.
- Provide sufficient semantics to specify the interface to repositories for the sharing and storage of model artefacts.

4.3.1 Evaluation of UML

UML is particularly used in the information and computational viewpoints of ODP. Although UML could also be applied to other ODP viewpoints, there is currently limited application to these. Since a large number of protocols with which IN needs to inter-work are already specified in ASN.1 and SDL, it might be considered for backwards compatibility purposes to use ASN.1 and SDL rather than UML in the engineering viewpoint.

5 Advantages of using Object Orientation for service modelling

Object oriented modelling is based on the principle of "objects". Objects are components that are self-contained, i.e. their properties can be described independent from the outside world. The definition of an object comprises attributes and methods. Attributes describe the data in the object; methods the operations that can be imposed on that data. Objects can invoke methods in other objects. Other very important concepts of object oriented modelling include inheritance and encapsulation.

Given the wide scope and the complexity of services envisioned for IN CS-4 (resulting from the integration of mobile, broadband and IP based network services) the advantages of using the object-oriented paradigm within the standardization of IN CS-4 are threefold:

- Object orientation methods seem suitable for specification of IN services throughout the whole design. However, it should be investigated whether IN service modelling really is an application area where OO methods can be successfully applied as well.
- Object oriented methods are already widely applied in software engineering.
- An object oriented Distributed Processing Environment (DPE) can be used as a uniform computational model providing transparent distribution for the network intelligence (service logic, service creation and management). A migration scenario could be the interworking between a CORBA-based DPE (for the network intelligence) and the existing IN equipment (SSPs) through a gateway (SS7-IDL) interface as illustrated in Figure 5:

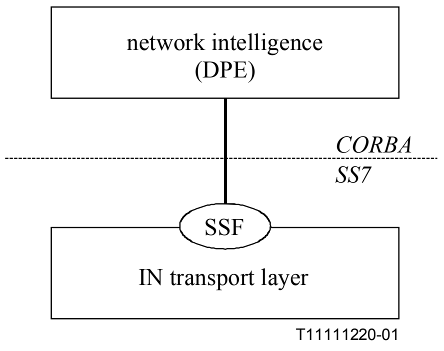


Figure 5 – A possible evolution scenario for IN

The advantages of using an object-oriented DPE (e.g. a CORBA-based platform) for the network intelligence include: enhancement of service reuse and generality (for composition and decomposition), ease of development, deployment and integration, dynamic binding and reconfiguration of service components.

Common Object Request Broker Architecture (CORBA) allows for invocation of object methods in a distributed environment. This means that CORBA will hide the location of the object that is addressed. The interfaces between objects specified with CORBA IDL provide technology transparency because the interface will hide how objects have been implemented. It is however worth noting that, currently, one of the major drawbacks for use of such environments in the context of IN is the real-time requirements!

The definition of interactions between objects across a logical interface is usually referred to as an API. Figure 6 illustrates the use of APIs within an IN context.

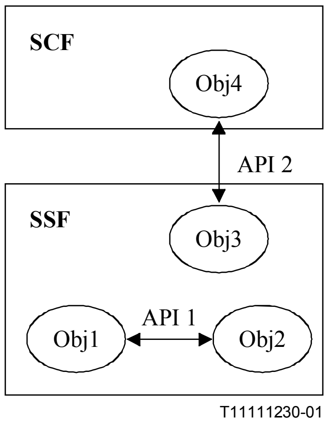


Figure 6 – An example of the use of APIs in an IN context

In Figure 6, Object 1 and Object 2 reside in the SSF. Object 1 could for example, be a Call Model Object and Object 2 the Trigger Status Table. When the SCF arms a particular Event Detection Point (EDP), Object 1 would invoke a method (API call) to Object 2 changing the status of a particular trigger. The API call could therefore be "arm edp2".

When Object 3 in the SSF invokes a method on Object 4 (API 2 call) residing in the SCF, the method crosses an external interface and as a result it **can** be mapped onto a protocol operation. The properties of the API call, i.e. its contents, will need to populate the physical protocol operation. Therefore a one-to-one mapping exists between the contents of the API call and the attributes of, for example, an INAP operation. For example, imagine Object 3 represents a "service access" object and Object 4 an IN service logic. Upon receipt of information from the user, Object 3 might invoke a method in Object 4 like "receipt of user dialled digits". This method could be realized as the INAP operation "AnalysedInformation".

5.1 Exploring the Use of APIs in IN CS4

In the evolving world of Telecommunications the need to keep up with new technologies and network concepts, particularly in the *Information Technology* area is as important than ever.

A common need across all of these architectures is the reuse of service scripts and the ability to send and receive information packaged in a recognisable form. Standardized and proprietary protocols work along side each other and often a need arises to map between one protocol and another at gateways or interworking mediums.

It has become much clearer of late of the need to describe services using a common format. Services in the IT domain are utilizing techniques associated with Application Programming Interfaces (APIs). International Standards particularly those associated with the Intelligent Network may benefit from this work.

5.1.1 Background

For any new service proposition, two questions must be asked, namely:

1) How will the service be supported on other transport technologies *(service-network interworking)*.
2) How will this service inter-work with other services *(service-service interworking)*.

Generally, the answers to these questions are not easy to find, but the exercise of asking them can result in much more generic, flexible and evolvable service definitions. For example, the definition of a voice mail service for the PSTN should be capable of running on the IP based networks and GSM networks, and should be capable of interworking with services such as personal mobility, e-mail, and video-telephony.

A component-based approach, using well-defined APIs, should be investigated for specifying and building services, such that improved service-network and service-service interworking is achieved.

Subclauses 5.1.2 to 5.1.4 outline a possible framework for the way in which a component-based approach to building telecommunications services could be used for the specification of Intelligent Network and protocols.

5.1.2 A Framework for the use of APIs

The framework is based upon the computing model of Client/Server. The client represents the entity requesting a function to be performed, the server represents the entity that performs the function and (optionally) returns the result. The client and server can be represented as components (collections of software objects or SIBs). The interface between the client and server is defined by the API presented to the client by the server, and is enabled by underlying middleware (e.g. transport protocols).

This approach is complementary to those methodologies already adopted for the Intelligent Network standards, and enhances them by taking a more software-orientated view particularly of the Distributed Functional Plane (DFP). The functional entities identified in the DFP make good candidates for the initial components to which the APIs are defined. For example, the information flows to and from the SSF/CCF provide a good basis for defining an API.

As more functional entities are defined (for example, bearer connection control for B-ISDN, voice messaging, service brokering), the interfaces to many of these capabilities will need to be standardized. The component-based approach allows these capabilities to be defined by their APIs in a way that they can be easily integrated from a software perspective. The APIs can then be used as the basis for developing protocols.

For complex components, these could be broken down further into objects and the protocol defined from these. For simple components, the protocol can be derived directly. By defining the API in detail, the API-protocol mapping is simplified and is more future proof. For example, it will be easier to evolve the current IN standards to align with distributed computing technologies.

Figure 7 shows how services and components are related. Each component is defined by its API – the API reflects the complete functionality of the component (including usage, management, etc.).

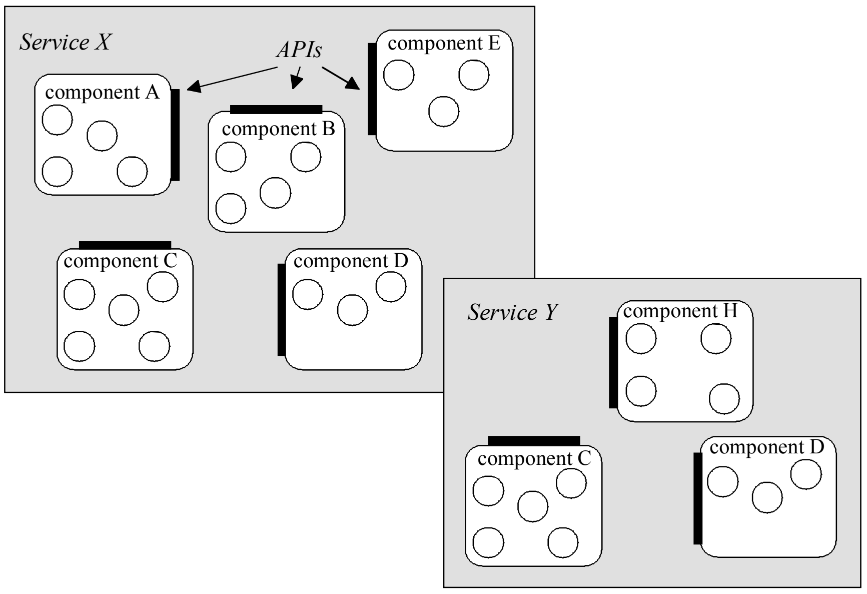


Figure 7 – Services/Components/APIs

5.1.3 API Overview

The API is the key to defining the client/server components. The API is the definition of the functionality of a component. It is essentially a set of operations (or methods) that can be invoked on the component, each of which causes the component to exhibit behavioural functionality. Each operation is specified syntactically as an identifier which identifies the operation being invoked, and parameters which affect the behaviour of the component in some way.

For example, the following API operation (*add_transaction*) on a billing component would cause the transaction identified by the *transaction_id* parameter to be added to the bill of the customer identified by the *customer_id* parameter:

```
add_transaction ( transaction_id, customer_id )
```

The API is abstract from the technology used to realize the component and is specified using an abstract interface definition language. In addition to the syntax, the API specifies the semantics of how functionality is invoked and the behaviour of the component, in terms of the way that public and private attributes are used and modified, and the results and errors returned. The API defines the complete set of functionality that a component can perform.

A particular presentation of an API represents the way in which the component is realized, and the concrete syntax and semantics for invoking a capability. The presentation may restrict the functionality that is accessible to an invoking component. A component may have a number of presentations, but only has one API. For example, the usage presentation of an SSF/CCF API would only include the call handling operations, and could be realized using INAP and/or ISUP protocols; the management presentation would support capabilities such as call gapping, and could be realized using INAP or CMIP protocols.

5.1.4 Example API for call processing

The call-processing component (SSF/CCF) has a number of potential clients: end users (referred to here as End Parties), service providers (referred to here as Third Parties), management, etc. In this example, only the End Parties and Third Parties are considered. The End Party differs from the Third Party in that it is the End Parties that request the calls and bearer connections to be established, maintained and released between each other. The Third Party can act on behalf of an End Party, or modifies the way that the call processing operates, with or without the knowledge of the End Parties.

Each 'API Call' (Operation) describes an operation made either by an End Party, or a Third Party. The API Call will be consistent between points A and B, although the underlying protocol and Network transport mechanisms may change at various points between the two. The following example (Figure 8) may clarify the point.

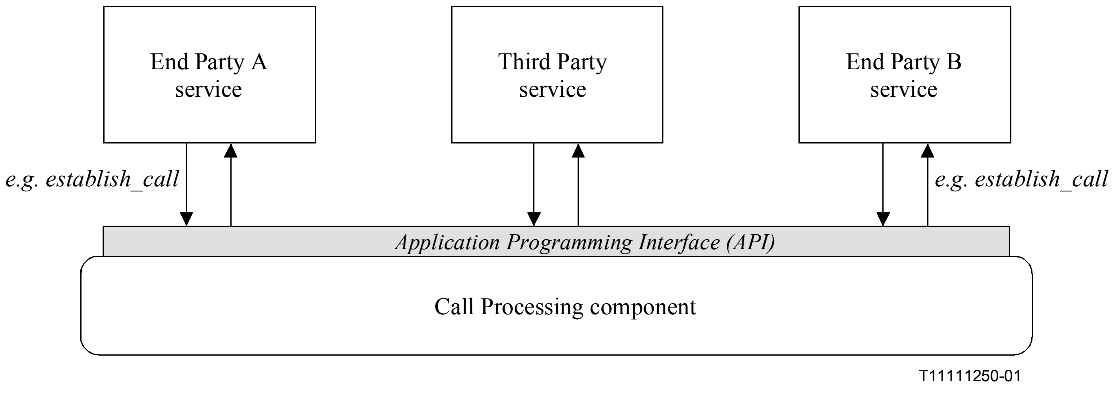


Figure 8 – Call Processing API example

Let us assume that an event (such as a telephone Off Hook) is detected by an End Party as an ISDN service (running in the terminal). The ISDN terminal is unlikely to have any relationship with the Network until the User dials a number. Once the End Party A service has collected enough digits from the user, it would then invoke the "Establish Call" API operation on the Call Processing component.

This would result in the necessary signalling being sent through the network between the various call processing component objects, for example using DSS1 and ISUP protocols. Once the call reaches the far end, the Call processing component will invoke an API call on a service component in the End Party B service, and this could be called "Establish Call", for symmetry.

If a more complicated scenario were envisaged, then the third party may register with the Call Processing for event notification when specific criteria are met. If the criteria are met, the Call Processing will invoke an operation on the Third Party service component which requested the notification.

It is quite easy to see how the Call Processing API to a Third Party could map onto existing IN standards (e.g. INAP protocol). It is less easy to map the End Party API, since this has not yet been addressed in the IN standards.

5.2 The SIB approach

This subclause seeks to highlight the shortcomings of using the IN SIB approach.

The breakdown of services into smaller reusable parts has served the IN reasonably well in the past, and is being used by many manufacturers as part of their Service Logic Execution Environments (SLEE).

The whole idea behind this methodology is to make as many blocks of data reusable and able to be executed across many service applications. The concept of service reusability is important and efficient and in reality may be implemented irrespective of how one models a service. However, when one compares the "SIB" methodology to Object Oriented methodologies, the shortcomings of SIBs are very apparent. The following is a list of those shortcomings:

- First of all it is a difficult process to refine (SIBs) at the (GFP) to the Distributed Functional Plane (DFP) level.
- The refinement from the Service Plane to SIBs on the GFP uses only one intermediate step, namely that of High Level SIBs. As a result, the refinement of the Service Plane to the GFP is not very precise. In order to achieve a more stepwise refinement, the concept of High Level SIB is not sufficient, but a more iterative refinement is required.
- Modelling the GFP by means of SIBs does not provide a complete service model since the service data is subordinate to the functions performed by an IN service. Data needs more explicit attention, e.g. to enable better management of service data.
- The SIB approach applied in IN CS1 and CS2 GFP modelling differs from modelling techniques applied in the IT world. However there is an increasing need to align products originating from the IT world (e.g. billing and customer care).
- The specification techniques used in the Global Functional Plane and the Distributed Functional Plane do not allow any mechanisms of consistency checking between specifications and thus limit service compositions and reusability. More generally, there is no global specification methodology advocated by IN.

6 A possible Evolution from SIBs to Object Oriented Service Capabilities

The gap between services and service features on one side and information flows and protocol on the other side is big. This implies that the completeness of the information flows and protocol operations/parameters is difficult to verify against the services and service features that need to be supported. In IN CS-1 and IN CS-2 the GFP was used to achieve a "layer" between the services and service features on one side and the information flows and protocol operations/parameters on the other side. This is a means to support validation of protocol completeness. In order to be able to model complex services the modelling techniques currently used on GFP need to be enhanced. Object orientation is considered a promising technique for service modelling. As a result, it is useful to investigate the evolution from the current modelling techniques to a more object oriented approach.

6.1 Service Class model

The Service Class model comprises two complementary views:

- *Service Execution View*. This view comprises the most elementary object classes that are available for service development.
- *Service Provisioning/Customization View*. This view shows the object classes that are visible during service provisioning and customization. It hides the object classes that cannot be managed while the service is in service.

These two views are both what is referred to as "object models" in several OO methodologies. This means that these views both provide a static view on the service. Some object classes can be accessed in the Service Execution View, others in the Service Provisioning/Customization View and some even in both views. This is represented in Figure 9. The instances of the object class "Call", e.g. contains data that changes during service execution. Instances of the object class Announcement contain data that may be read during service execution (playing an announcement). The content of the announcement data may be changed with the service management system. Therefore, the

Announcement object class is in the intersection of the Service Execution and Service Provisioning/Customization View.

Figure 9 distinguishes between Service Resources and Network Resources. An example of a Network Resource object class is the Terminal object class. The relationship between the Service Resources and Network Resources expresses how service capabilities are mapped onto network resources. For example, a user can make a call from different terminals such as his POTS terminal or ISDN terminal. The object classes in the Service Execution View of which instances are created during service execution are *dynamic* (e.g. Call), all the Service Resources are *persistent*, since their lifetime is longer than a single service execution.

For service modelling, only the Service Resources are relevant. These object classes are the service capabilities used to compose services.

Note that the object classes in Figures 9 and 10 are just examples.

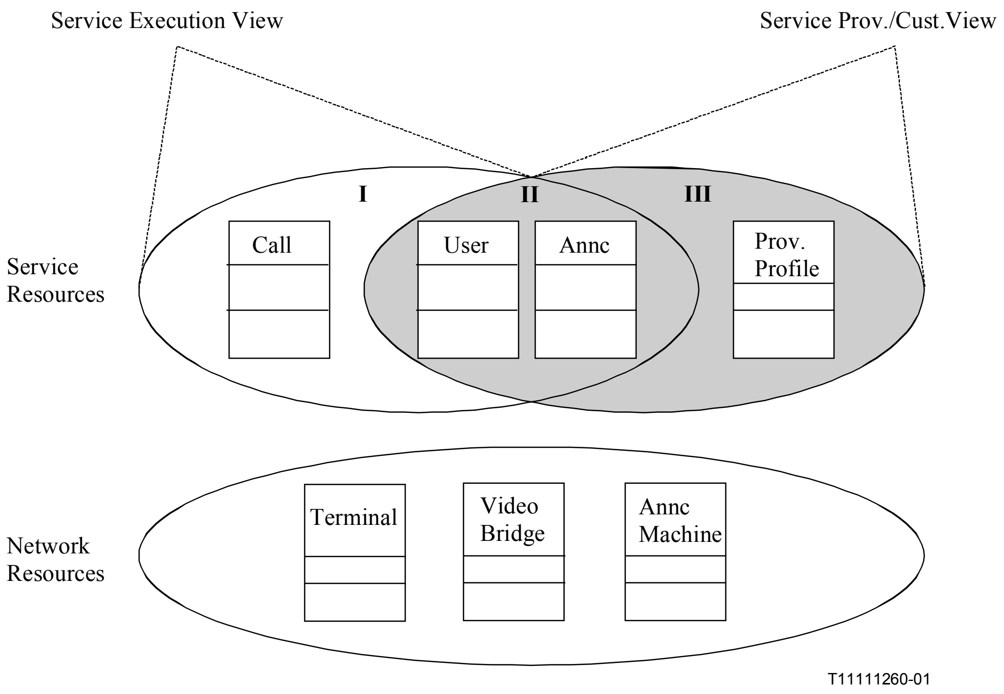


Figure 9 – Different Views on a service

6.2 Service Execution View

Figure 10 shows the object model of the Service Execution View at the highest level. The *Service Logic* object class represents the service and may consist of zero or more (0+) *Service Logic* object classes and zero or more *ServiceResource* object classes. This allows for modelling a service as a composition of service features (more general, it allows for stepwise refinement). At the lowest level of decomposition, we see the *ServiceResource* object classes. The *ServiceResource* object classes have a service independent character and can be considered the object oriented equivalent of the SIBs presently used. A *ServiceResource* object class may be reused in zero or more *Service Logic* object classes (0+). An instance of the *ServiceResource* object class is either a *DynServResource* or a *PersServResource*. The *DynServResource* object class represents all object classes of which the object instances exist only during a single service execution (see I in Figure 9). The *PersServResource* object class represents all object classes of which the object instances are persistent, i.e. their lifecycle is longer than just a single service execution, e.g. the period of time a customer subscribes to a service (see II in Figure 9). The *ServSessionMgr* invokes the service and could be imagined as the Basic Call Process in the IN CS-2 GFP. The *ServSessionMgr* itself can be seen as a specialization of the *DynServResource* object class, but is depicted separately because of its special role.

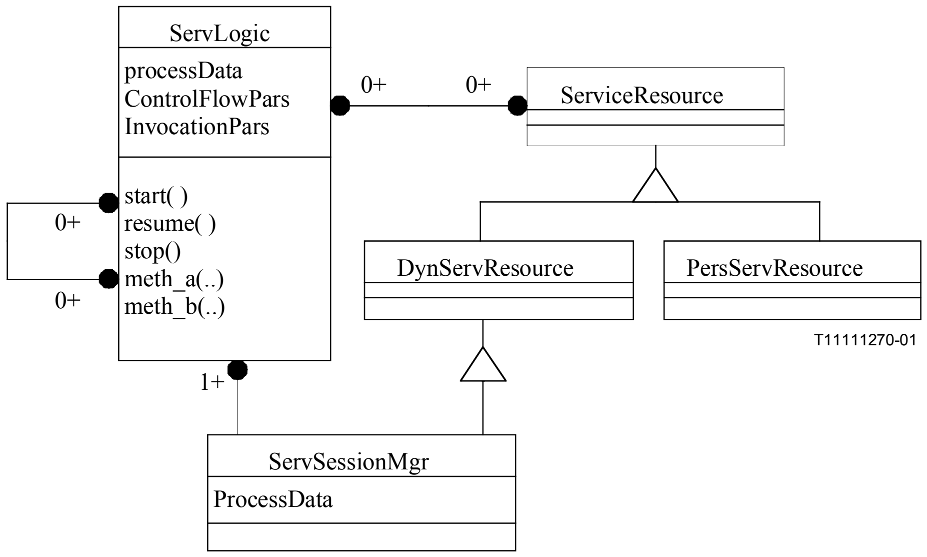


Figure 10 – The object model of the Service Execution View

The *DynServResource* object class and *PersServResource* object class can each have zero or more specializations. A specialization of the *DynServResource* object class could be the *Call* object class (see Figure 11). An object instance of this class can contain, for example, call instance data such as the A-number and B-number. Methods of this class could be Add_Party( ), Remove_Party( ) and Connect( ). Possible specializations of the PersServResource are, for example, *Announcement* and *Counter*. These object classes are persistent since they exist longer than a single service execution. They are viewed in both the Service Execution view and Service Customization/Provisioning view, since methods in object instances of these classes can be invoked in both views. For example, during execution of the service logic the data of an *Announcement* object can be read; during service customization the text of the announcement can be modified.

6.3 Migration of CS-2 SIBs to Object Classes and Methods

In order to achieve an evolution from the service modelling techniques currently used towards a more object oriented approach, it is useful to map the SIBs onto object classes and object methods. The mapping depicted in the table following Figure 11 shows how this could be done.

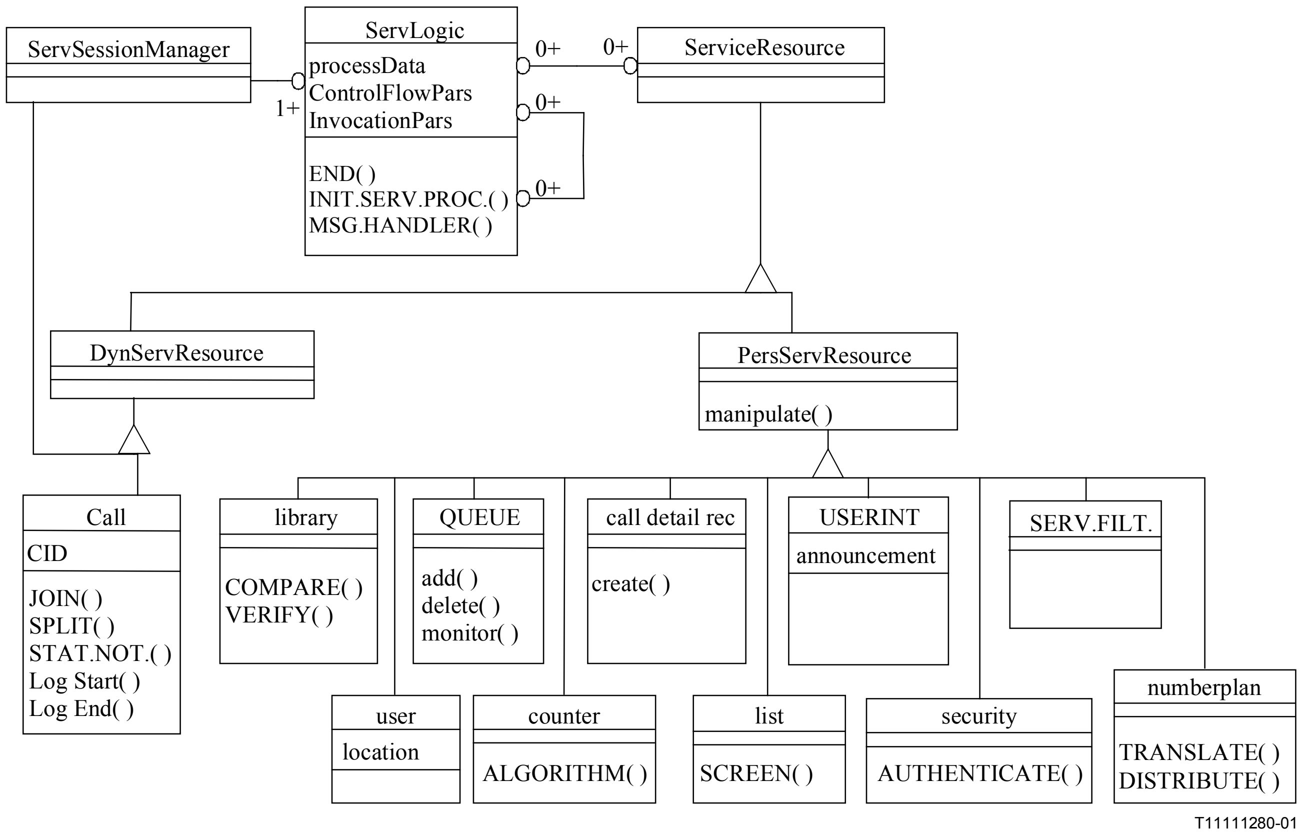


Figure 11 – A possible migration from SIBs to object classes and object methods

The table below describes the mapping of the IN CS-2 SIBs (Q.1223) into the Object classes used above:

SIB	Mapping
Algorithm	Applies a mathematical algorithm to a value (such as incrementing or decrementing a counter) and can therefore be seen as a method of the *Counter* object class.
Authenticate	Function with security purposes and therefore can be considered a method of the *Security* object class.
Charge	Charging can be seen as the creation of a call detail record and therefore represented as the *Create* method in the *Call Detail Record* object class.
Compare	Function used to check an identifier against a specified reference value, such as time of day, originating location or digit value. It can be seen as a generic mathematical operation and it is therefore represented as a method in the *Library* object class (which can be considered a standard library with mathematical and string operations as found in many programming languages).
Distribution	Distribution of a call to a destination according to a particular algorithm can be seen a method in the *Numberplan* object class.
End	Indicates the normal termination of a service process and can be seen as a method of the *Service Logic* object class.
Initiate Service Process	Invocation of parallel service logic can be seen as a method of the highest level object class, i.e. the *Service Logic* object class.
Join	Attaching a call party or call parties from the current call group to another call group of the same call can be seen a method of the *Call* object class.

SIB	Mapping
Log Call Information	Logging of identified call instance data is seen as the methods *Log Start* and *Log End* in the *Call* object class, in which call instance data are attributes.
Message Handler	Communication between processes in a single service can be seen as a method of the highest level object class, i.e. the *Service Logic* object class.
Queue	A *Queue* object instance contains zero or more references to *Call* object instances. *Call* instances can be added, deleted etc. to, from the *Queue.*
Screen	Checking a number against a list of other numbers (e.g. for call screening) can be seen as a method of the *List* object class.
Service Data Management	Performs operations on service data, such as add, delete, modify, retrieve, etc. These operations are collected in the *PersServResource* object class as the *Manipulate* method.
Service Filter	Filters calls according to a specified mechanism. It is considered a separate object class with methods such as *Activate* and *Report.*
Split	Detaches a call party or group of call parties from the current call and attaches the party/parties to a new or other already existing call. This could be a method of the *Call* object class.
Status notification	Provides the capability of inquiring the status (changes) of network resources. It can be seen as a method in the *Call* object class, allowing for status checking of the network resources that enable the *Call.*
Translate	Translation of a number to another number could be a method in the *Numberplan* object class.
User interaction	The prompt and collect mechanism to provide the user with information and to achieve information from the user is considered as a separate object class.
Verify	Can support syntax checking of all kinds of data and is therefore represented as a method of the *Library* object class.

APPENDIX I

Bibliography

[TINA] TINA-C architecture specifications (e.g. Service Architecture and Information Architecture).

[OMT] *Rumbaugh et al.*: Object-Oriented Modelling and Design.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	**Switching and signalling**
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2001