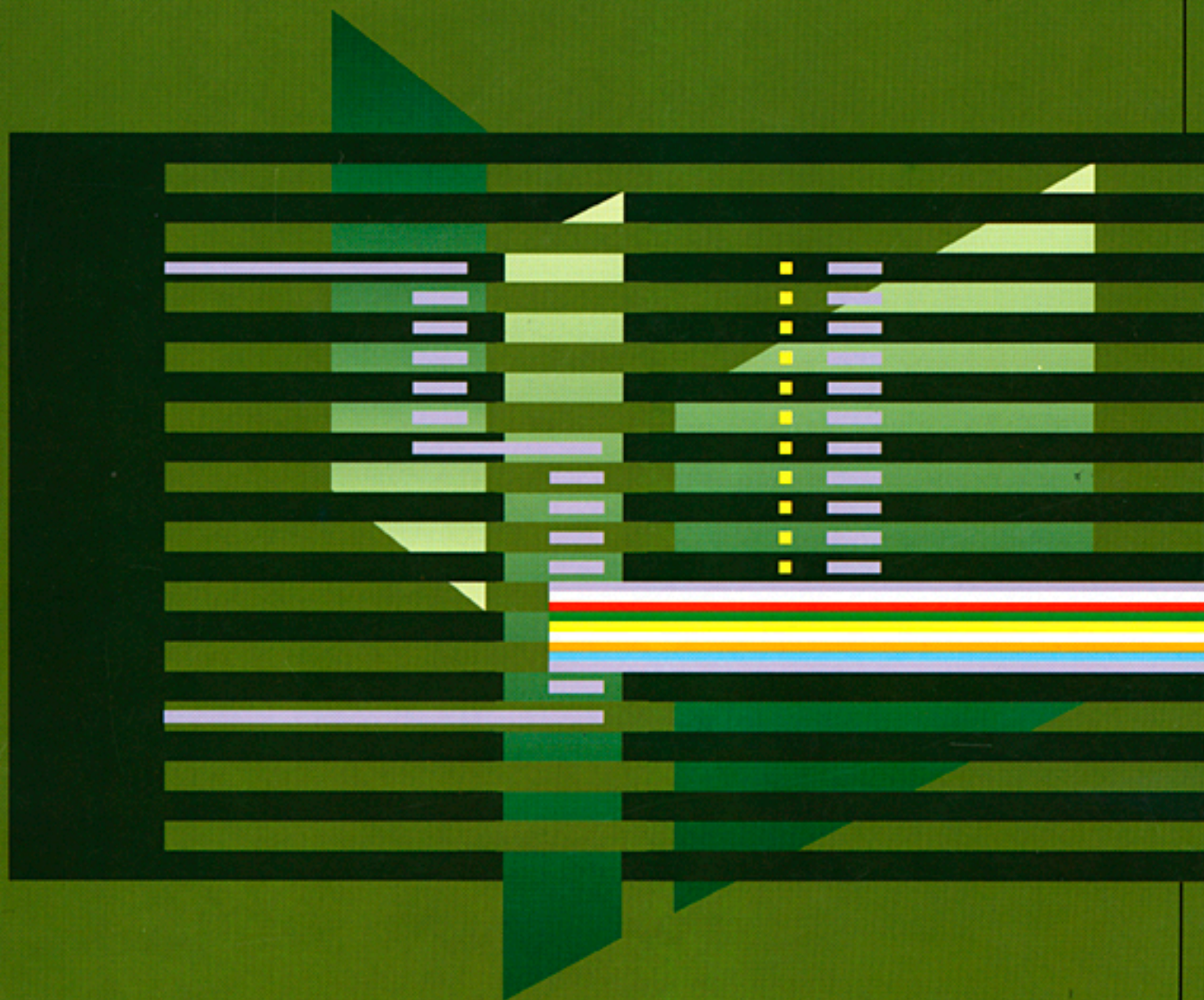


## Intelligent Networks





# Innhold

Guest editorial <i>Bjørn Løken</i>	3	IN control of a B-ISDN trial network <i>Kirsti A Løvnes, Frank Bruarøy, Tom Handegård and Bengt G Jensen</i>	52
An introduction to Intelligent Networks <i>Jan A Audestad</i>	5	Some issues of security and privacy in Intelligent Networks <i>Klaus Gaarder</i>	67
The Intelligent Network service life cycle <i>Håkon Vestli and Raymond Nilsen</i>	15	Artificial intelligence in the network: A rule based service control system prototype <i>Harald Seim</i>	73
How to handle all the services <i>Harald Seim</i>	26	Universal Personal Telecommunication and Intelligent Network architecture <i>Jan Audestad and Børge Jacobsen</i>	79
Description and specification of services in Intelligent Networks <i>Erik A Colban and Klaus Gaarder</i>	35		
Standardisation activities in the Intelligent Network area <i>Endre Skolt</i>	43		

**Teletronikk**  
Televerkets tekniske tidsskrift

**88. årgang Nr. 2 - 1992**  
ISSN 0085-7130

**Redaktør:**  
Ola Espvik  
Tlf. + 47 6 80 98 83

**Redaksjonssekretariat:**  
Gunhild Luke  
Tlf. + 47 6 80 91 52

**Adresse:**  
Teletronikk  
Teledirektoratets forskningsavdeling  
Postboks 83  
N-2007 Kjeller

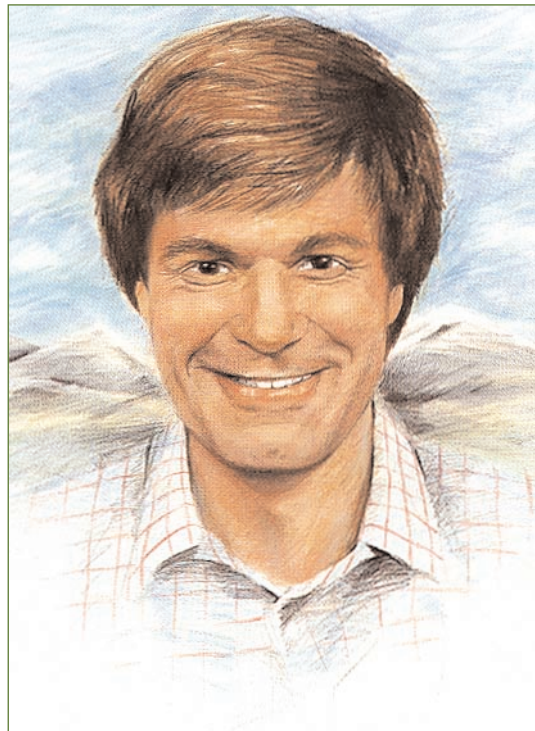
**Redaksjonsråd:**  
Teknisk direktør Ole Petter Håkonsen  
Forskningsdirektør Karl Klingsheim  
Forskningsjef Bjørn Løken



Teletronikk utkommer med  
fire nummer pr. år.

# Guest editorial

BY BJØRN LØKEN



With mobile telephone systems like NMT we have become familiar with the flexibility and ease of use of automatic roaming. With automatic roaming the subscribers can access the network where they happen to be at the time of call. An exchange in a mobile network must therefore be able to serve any subscriber entering its service area.

Such flexible service handling is not possible in existing fixed networks because call control for a subscriber is permanently associated with a particular exchange, i.e. the exchange where his subscriber line terminates. However, one of the near term targets of service development is to extend mobility to fixed networks by introducing personal mobility.

The introduction of mobile services has necessitated a change in traditional network design. In order for an exchange to give services to all roaming subscribers, subscriber data must be easily retrievable from sources external to the exchanges. This leads to the requirement that call control and subscriber data must be functionally separated. In systems with personal mobility also the service itself must be executed in separate network nodes.

Services such as advanced freephone, televoting, credit card calling and virtual private networks will impose similar requirements on the network architecture. This enhanced architecture is called Intelligent Network (IN).

The original idea behind IN was that certain services were more efficiently implemented by allocating part of the service execution to separate network entities (Service Control Point). For other services this was the only way in which they could be implemented. One important point is that mobility and IN represent two routes towards the same type of network architecture.

An implementation of a network where a separation between call control, service control and user data is introduced requires the support of a signalling system capable of supporting real time transaction oriented exchange of information between network nodes dedicated to switching, data storage and service processing.

The transaction Capabilities Application Part (TCAP) of Signalling System No. 7 has been developed for meeting these requirements. It is also the first step towards a vendor independent application layer interface between network entities.

In order to obtain harmonised IN services internationally standards must be developed. A first set of such standards will be put forward to the CCITT Plenary Assembly in 1993 for adoption as the Q.1200 series of Recommendations. The Q.1200 series includes network architecture, a set of capabilities or building blocks for service creation and allocation of functions to physical entities with associated protocols. These protocols are the second step towards a vendor independent interface.

Looking at the way this international work on IN has taken one may say that IN is not a specific technology but rather a methodology for future network development.

Intelligence in the network means that services are separated from basic switching and transmission.

When IN has been fully developed, it will offer several benefits to all parties involved: network operators, service providers, service subscribers and users.

Some of these are:

- rapid introduction of new services
- reuse of software and software portability
- vendor independence
- customisation of services
- subscriber control of service attributes
- one stop shopping
- personal mobility.

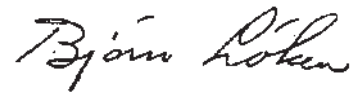
In order to exploit the benefits much research is required on methods and tools for

- service specification
- service creation
- service testing and validation
- service deployment.

Efficient solutions in these areas will be an important element in an environment where network operators are competing for gaining market shares. Those who can deliver what the users need at the right time at an acceptable price will be the market winners. IN evolution is thus one important strategic element for network providers in the future. Many network operators are

beginning to see this and are, therefore, putting much resources into IN research. Since this research is expensive and requires much resources, several network operators have joined forces, for example in EURESCOM and RACE, even though they may become competitors in the future. Most of them expect to gain more from cooperation in research than they may eventually lose from competition. Norwegian Telecom is also participating in this work. The advantages for us is that, being a small network operator, we gain access to results that we otherwise could not afford to produce on our own.

The research will also determine the direction in which IN standards develop and will in the long run replace the proprietary standards the network operators are forced to buy from industry today.

A handwritten signature in black ink, reading "Björn Roken". The signature is written in a cursive style with a large, prominent initial 'B'.

# An introduction to Intelligent Networks

BY JAN A AUDESTAD

## Motivation and objectives

Until recently telecommunications services in public networks were limited to basic calls, that is to establishment of connections between two users, with the addition of a few simple supplementary features. It is also recently that much of the electro-mechanical equipment were replaced by modern program controlled exchanges interconnected with Signalling

System No. 7. The exploitation of the vast computing power of these exchanges and the capability of Signalling System No. 7 to pass service related information have led to the development of a large number of new basic services such as mobile services, enhanced freephone services, virtual private networks (VPN), universal personal telecommunications (UPT), CENTREX, etc. In addition, the

number of supplementary services, i.e. services or features that can be combined with basic services to meet specific user needs, has grown from a handful to more than a fathomful. At the moment the overall picture of the capabilities the networks can offer is complex and it is difficult for users both to choose the right services and to use them in an efficient manner. One motivation for IN is to pro-

621.39.05  
681.324  
681.3.01

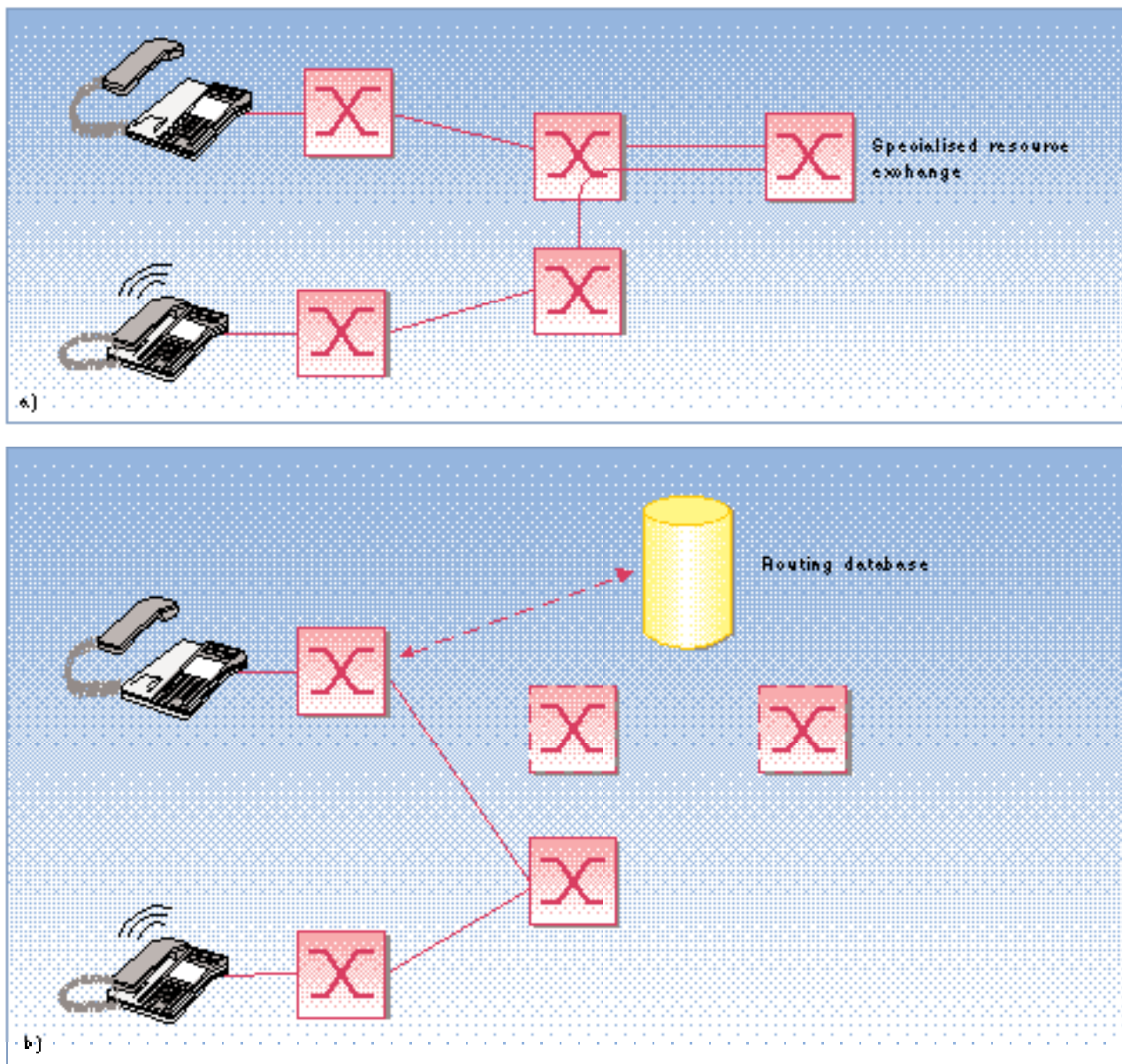


Figure 1 Handling of simple freephone service

In non-IN networks the conversion of freephone numbers to physical address takes place in a specialised resource exchange (a). The physical route of the call will be long and complex. In IN (b) the outgoing exchanges can interrogate a routing database in order to obtain the physical address.



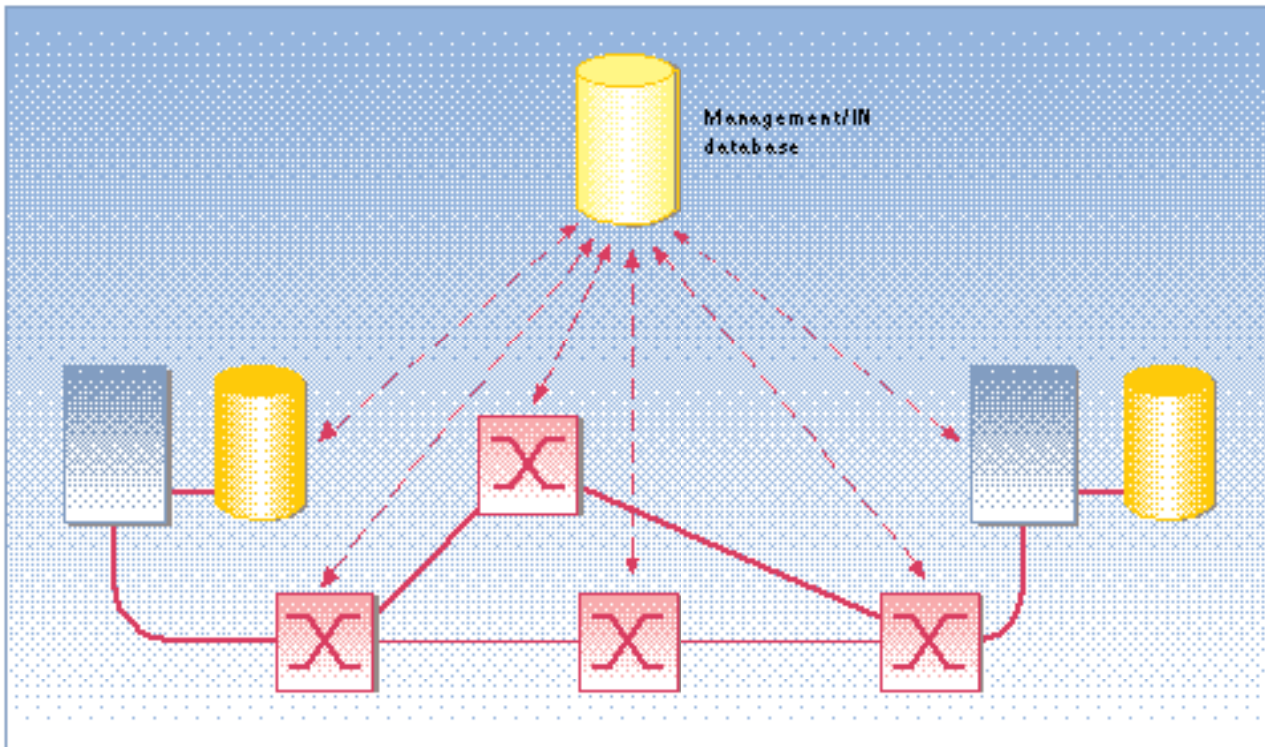


Figure 2 Handling large volume of traffic

By supervising the available traffic capacity on a number of paths between two subscribers, the management/IN database can determine when there is sufficient capacity and then take actions to establish a suitable path.

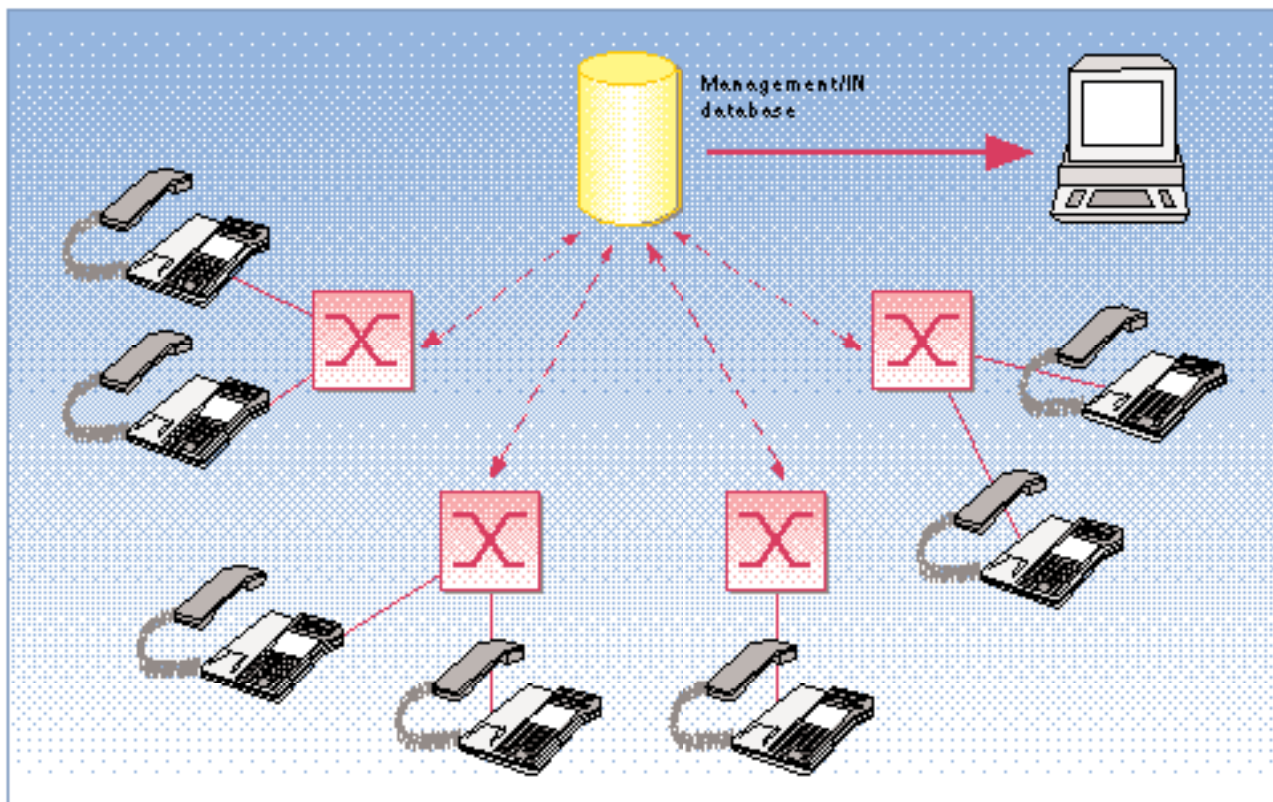


Figure 3 Televoting

The votes are counted by the local exchanges and subsequently passed to the management/IN database where they are collected and formatted in a suitable form for the customer.

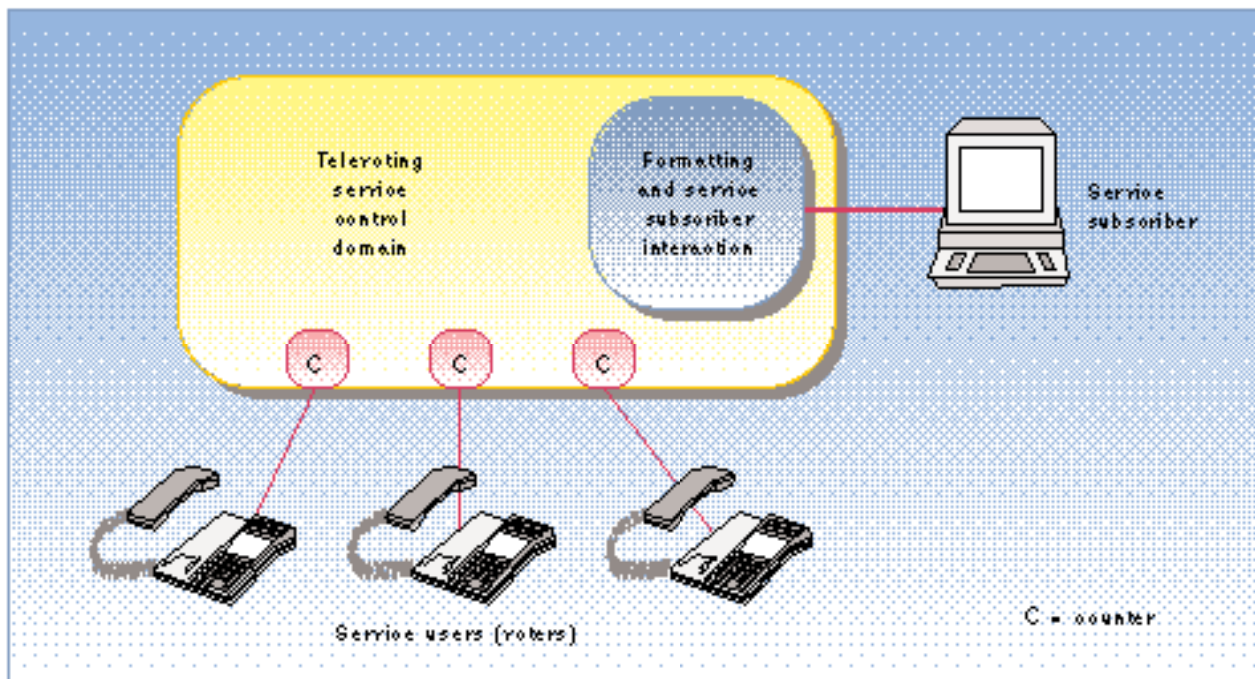


Figure 4 Another view of televoting

In this view the service users and the service subscribers interact directly with the service execution function. The fact that the latter is distributed over databases, exchanges, etc., is not important for description or understanding of the service.

vide solutions to these problems. Another closely related motivation is to provide mechanisms for rapid and efficient adaptation to market demands for new services and features.

In order to offer the basic service capabilities mentioned above in an efficient manner the network needs to be enhanced with additional infrastructure. One objective of IN is to provide this infrastructure. The basic services are, in a broad sense, independent of the network that supports them, i.e. whether it is ISDN, broadband network or packet data network. The same applies to supplementary features.

This paper will look at various aspects of IN and explain them by help of examples. In this presentation the main focus will be on principles rather than on current standards and early implementations in order to illustrate the evolutionary potential of IN. Current standards and implementations are devoted to practical adaptation of the basic principles to existing network implementations and equipment design and do not take full advantage of the capabilities an IN may offer.

The exposition of this paper is based on some of the key elements in the definition of IN adopted by the CCITT and included in Recommendation Q.1201.

These are:

- efficient use of network resources

- subscriber control of service attributes
- reusability and portability of ideas, methods and software
- flexible allocation of functions to physical entities
- integration of all aspects of management and administration of services.

The term subscriber will be used to designate the person or entity having the contract with the service provider. The user designates any party accessing the network using own or other services.

### Efficient use of network resources

In order to illustrate the versatility of IN to provide efficient network solutions, three quite different examples will be given: basic freephone service, traffic volume booking and televoting.

The main characteristic of the freephone service is that the calls are not paid by the calling user. The freephone number, i.e. the number dialled for accessing the network, is normally a number that alone does not identify the physical destination of the service. Reasons for this are:

- it should be possible from the dialled number to identify it as a freephone number. Therefore, it is common to allocate a general countrywide number series for this service
- the destination of one freephone number may be conditional, depending on

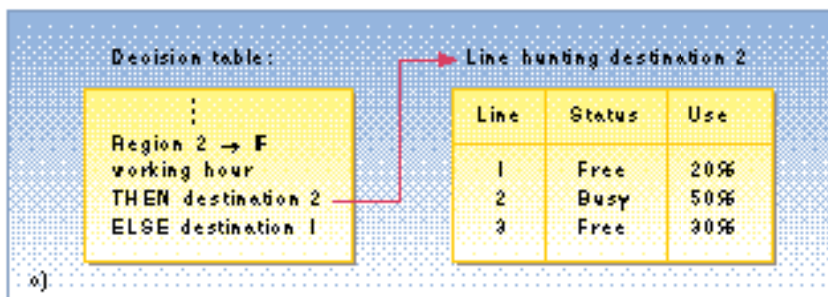
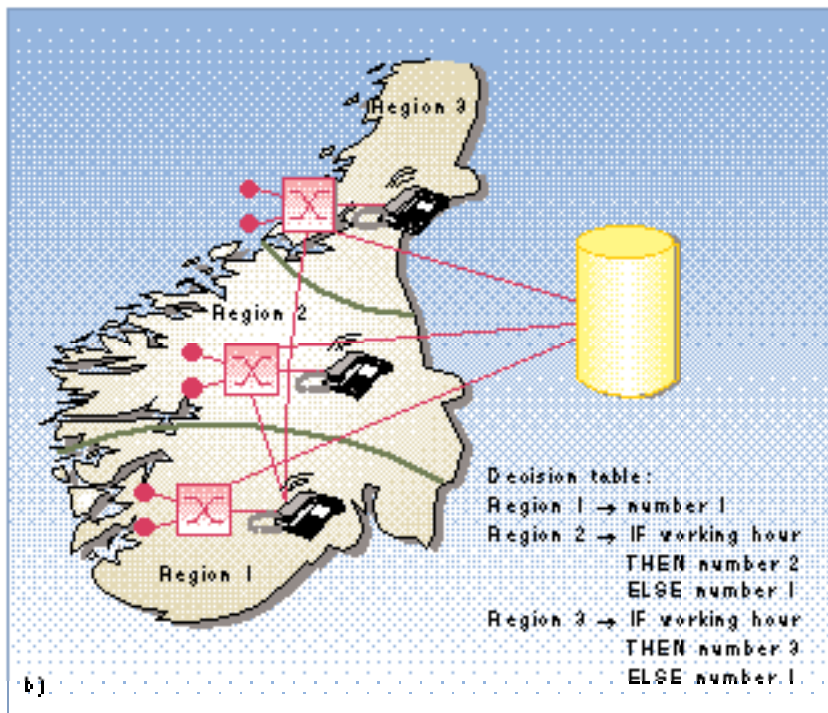
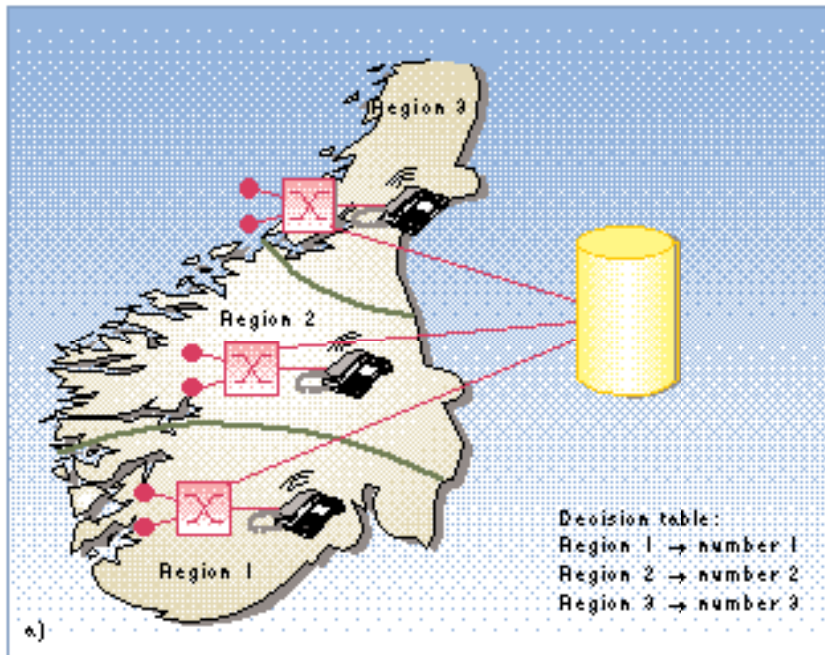
time of call, call origin, etc. An example of such freephone service is given in the next section.

Therefore, a translation from the freephone number to the physical destination number must take place in the network. As illustrated in Figure 1a) this number translation can be done by a special resource exchange without the need for IN. All calls to freephone destinations must then be routed physically via this exchange. However, by using IN, i.e. by introducing a routing database which can be interrogated by all exchanges in the network, a more efficient routing requiring fewer connection segments is obtained (see Figure 1b)).

For the simple freephone service the database can be viewed as a look-up table consisting of two columns, where one column consists of all freephone numbers and the other contains the corresponding physical destination numbers. When receiving a request for number translation containing the freephone number to be translated, the database will find the corresponding physical destination number and return it to the exchange for further call processing.

This example introduces one aspect of IN (which has more or less become the trademark of it), namely the use of databases and other equipment external to the exchanges. As we shall see later, this facet of IN leads us towards a better understanding of service execution and service based network architectures.





In the next and later sections we will return to the freephone example and show how IN allows the service to be enhanced in order to meet special subscriber needs.

The service of the second example can hardly be realised without some kind of centralised function which can supervise and take actions on the whole network. Figure 2 shows a case where customers (e.g. large computer installations) can request the network to provide capacity for transfer of huge volumes of data. The centralised function (management/IN database in the Figure) will monitor the available capacity of switches and transmission segments in order to decide when and on which route to offer information transfer.

The service resembles network traffic and routing management which is one of the capabilities of TMN (Telecommunications Management Network). It uses functions required for that purpose: determination of free capacity and available routes. The service is thus an example of using features required for administration and operation of networks as an element of a service offered to subscribers. It may also be taken as an indication that much commonality can be expected between TMN and IN.

The key feature of the service is efficient utilisation of the transmission capacity of the network. Variations of it could be:

- booking of capacity where the requested capacity must be made available at the required time
- selling capacity cheaply at low traffic hours
- select routes with desired characteristics
- etc.

The third example is televoting. Televoting can be used for elections, opinion polls, melody contests, quiz programmes, etc. Though quite different, all these applications share some common attributes that the network must support:

- ability to count votes on different voting alternatives
- ability to provide instructions to the voters
- ability to present results to the service subscribers.

Figure 5 Evolution of a freephone service



Televoting is, like freephone, a service which has been offered for a long time and without IN support. In non-IN implementations the calls representing voting events are routed towards special terminations in the network for registration, counting and processing. Since televoting often causes mass calling events with the risk of overloading the network, an IN solution like the one shown in Figure 3 is preferable. In this solution votes are counted in each local exchange and subsequently transferred to the management/IN database for further processing. This processing may consist of formatting the results in a way convenient to the service subscriber and delivered at times determined by him. The local exchange may also indicate to the voter that the vote has been registered or rejected, as the case may be. The indications to be provided may be specific to the type of televoting service used or in accordance with the wishes of the service subscriber.

Overloading of the network is avoided since calls exist only on user access lines and counts can be transferred from the local exchanges to the management/IN database at a rate that will neither overload signalling links nor processors in the database.

The example brings forward several points which are also key characteristics of IN even though they are not exploited in the first phase of implementations. The database contains information which has been determined (and also inserted) by the service subscriber. He can also alter some of the data sets associated with the service within given ranges. This point will be explored further in the next section.

The service is distributed in the sense that the central database and processing entity, though in control of the service, performs only part of the service execution: formatting and presentation of voting results and interactions with the service subscriber. The parts of the service related to registration of votes and interactions with the voters are decentralised to local exchanges.

The service subscriber interacts directly with the central database. The interaction between the service users (voters) and the service can, in a similar sense, be regarded as being direct. The televoting service may then be depicted as shown in

Figure 4. In this view the service is seen as a single entity, although it is physically distributed. The service does not require connection segments for passing information between users since there is no such information to pass. As we shall see later, this is part of a more general distributed IN architecture where service control and the transmission and switching aspects of the network are separated.

The main target of this section was to show how IN can be exploited in order to make efficient use of network resources. However, by giving three simple examples of IN services where efficient use of resources has been a prime factor in developing them, several other key aspects of IN have been uncovered:

- that there exists a close relationship between IN and TMN
- that there is a potential for allowing service subscribers the capability of manipulating certain service attributes
- that the service execution requires distributed processing and that this processing may be described in terms of an architectural concept in its own right.

### Subscriber control of service attributes and customisation of services

In the televoting example given above we saw that there are service attributes that the service subscriber may want to control. For a televoting service used for opinion poll such attributes could be:

- the region where the voting is to take place
- the number and type of voting alternatives
- the form in which the results are to be presented
- at what time and by which actions results are to be presented: intermediate results, final results, at given times, by request from the service subscriber, etc.
- the start time and duration of the voting

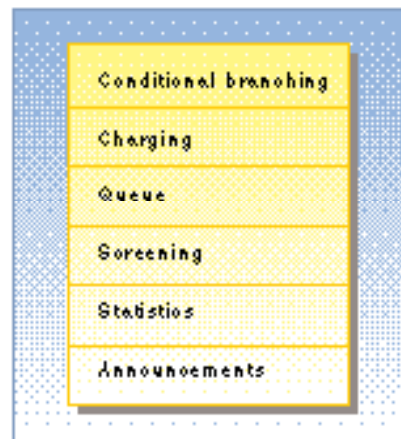
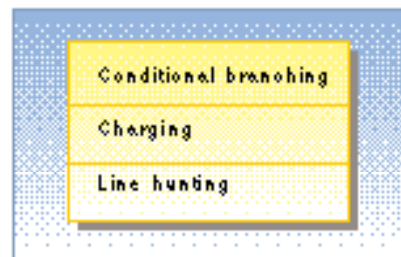
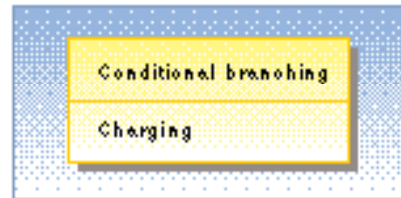
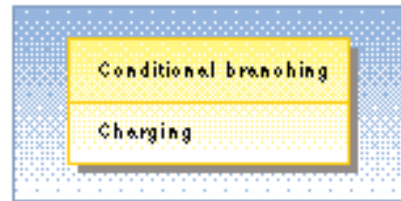


Figure 6 Service objects required at different evolutionary steps of the freephone service

- how to present the voting alternatives and other instructions to the voters
- who are allowed to vote
- how acceptance or rejection of votes is to be presented to the voters.

The set of attributes required for a simple melody contest could be the start and stop time of the contest, all other aspects of the service being fixed by subscription.

This example illustrates several things:

Object	Attributes
Conditional branching	Origin Time of day Day of week Date Terminal type Terminal characteristics ⋮
Queue	Maximum queue length Current queue length Number of servers Call treatment Identity of servers ⋮
Statistics	Number of calls Call duration file Call events file ⋮
Announcement	Information reference Information string ⋮

Figure 7 Example of objects and attributes

- The type of attributes being available for subscriber control will depend on individual requirements.
- The service as viewed by the service subscriber will look completely different from application to application: there is almost no commonality between the opinion poll service and the simple melody contest service from the service subscriber point of view even though much the same mechanisms are required in the network for performing the two services.
- The way in which the service subscriber interacts with the service may take different forms depending on the number and type of attributes being available for manipulation. These interactions may be anything from complex menu driven procedures to the use of simple push-button equipment.

The above brings forward another key issue of IN, namely customisation of the interaction between service subscribers and the network. As we shall see later, customisation is a wider aspect than this and enters into all facets of IN.

In order to give a more systematic approach to the issue of subscriber control of service attributes let us return to the freephone example and expand that in steps. In its simplest appearance this service consisted of

- levying the charges against the service subscriber
- converting the dialled freephone number to the number of its physical access point in the network.

Note that it is specifically stated that the charges are levied against the service subscriber and not against the subscription associated with the physical access (or the called number). As will soon become apparent, these subscriptions may be independent from a service point of view.

Our service subscriber wants to expand his business to three offices distributed over the country with the arrangement that

- the same freephone number is used for all the offices
- each office shall cover a given region of the country.

In order to support this service the network must be able to convert the freephone number to different destination numbers depending upon the origin of the call. Note that the simple conversion table has now become more complex since it is conditional by the origin of the call. The situation is illustrated in Figure 5a).

The service subscriber then wants to offer 24 hour service and service on holidays with the condition that only one office is open outside normal working hours. He also wants control of the attributes associated with time (time of day, weekdays, dates) and definition of geographic region. The new scenario is shown in Figure 5b) together with the algorithm for choosing the right destination.

The business has now expanded so much that there are several persons answering calls at each office. The service subscriber therefore wants line hunting in order to choose free positions and ensure equal loading of them. The line hunting feature requires that the service control function not only does number conversion but also that it keeps a continuous overview of which lines are free or busy within each hunt group (see Figure 5c)) and how incoming calls are distributed among them.

In the final step the line hunting feature and the conditional routing on time has been replaced by a centralised queue where the queuing function supervises which lines are free or busy and which offices are open or closed. The conditional number conversion arrangement of step b) still applies with the exception that if all lines are busy at one office, the call can be routed to another office even though that office is outside the region normally served by that office. The service subscriber has also changed the charging arrangement such that calls from a given group of customers to the business always have calls free of charge while other customers are charged if they call outside normal working hours. The service subscriber may also need customised announcements to be inserted at different call states. The announcements could be indication of how many calls are in the queue and expected waiting time, indication that the call will be charged to the customer, etc. He also requires that the network provides the



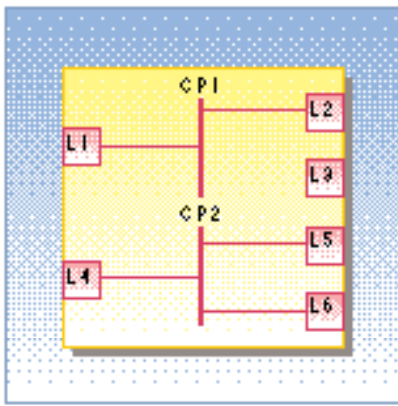


Figure 8 Virtual switch object

announcements to the calling user in a convenient way (voice, text, picture) depending on the type of terminal used. The service subscriber wants detailed monthly statistics: number of calls, handling times, line loadings, queuing times, number of unsuccessful calls, etc. He also requires control of most of the attributes associated with his service and even asks for customised security mechanisms related to authentication, authorisation, rules for invocation of changes, status of attribute values, etc.

What this example illustrates is a realistic evolution of a service after it has been installed in its first simple form. The real challenge for network operators and service providers is to be able to respond quickly to such demands for customisation of service. Customisation of services and rapid response to market demands require several prerequisites:

- methods and tools by which services can be quickly created on customer demands
- efficient deployment of the service into the network once it has been created
- responsiveness of the network to quickly adopt changes to the service in order to meet new customer demands.

## Reusability

The evolution of the freephone service given in the previous example hints to a possible decomposition of services into reusable modules, where “reusable” means that each module can be used in different types of service. This is the key to service creation and efficient service execution.

In what follows we will refer to these service modules as objects in order to flag that they should obey all general aspects of object oriented modelling and design.

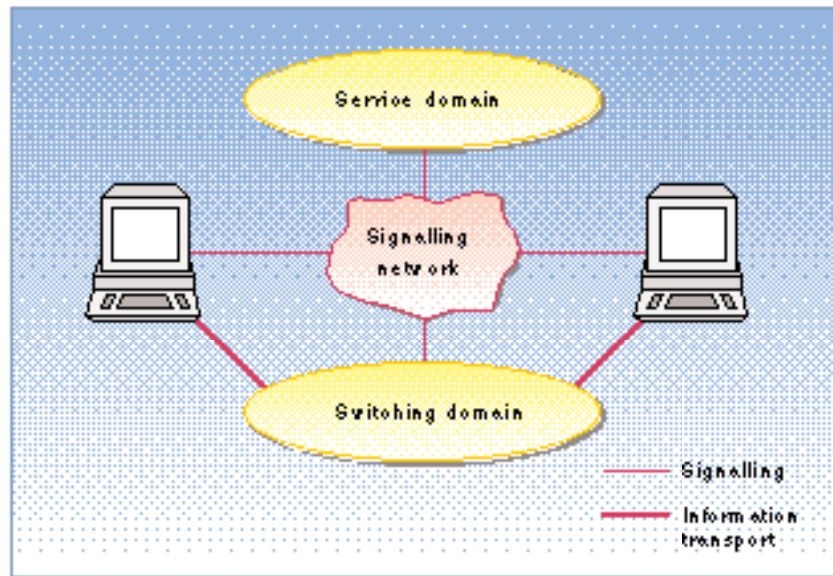


Figure 9 Service driven network architecture

From the previous discussion we may identify several service constituents that may serve as reusable service objects:

- conditional branching, where the condition can be any combination of call origin, time of day, day of week, date, terminal type, terminal characteristics, etc
- statistics
- screening
- queue
- line hunting
- counter
- authentication
- authorisation
- charging
- announcement.

By combining one or more of these objects a complete service can be built. Figure 6 lists which objects are required for each of the four evolutionary steps of the freephone example of the previous section. The list of objects is not enough to characterise the service. It is also necessary to specify exactly what each object is expected to do. This can be done by defining a set of attributes for each object as shown in Figure 7. Each attribute can again be defined in terms of which operations can be performed on it. The current queue length attribute of the queue object can be increased or reduced by 1 depending on whether a new call arrives or a queued call is being serviced.

The above description is not formally exact. It is intended only to give a first idea on how object orientation enters into service design. Accompanying papers provide further insight into the formal use of object orientation.

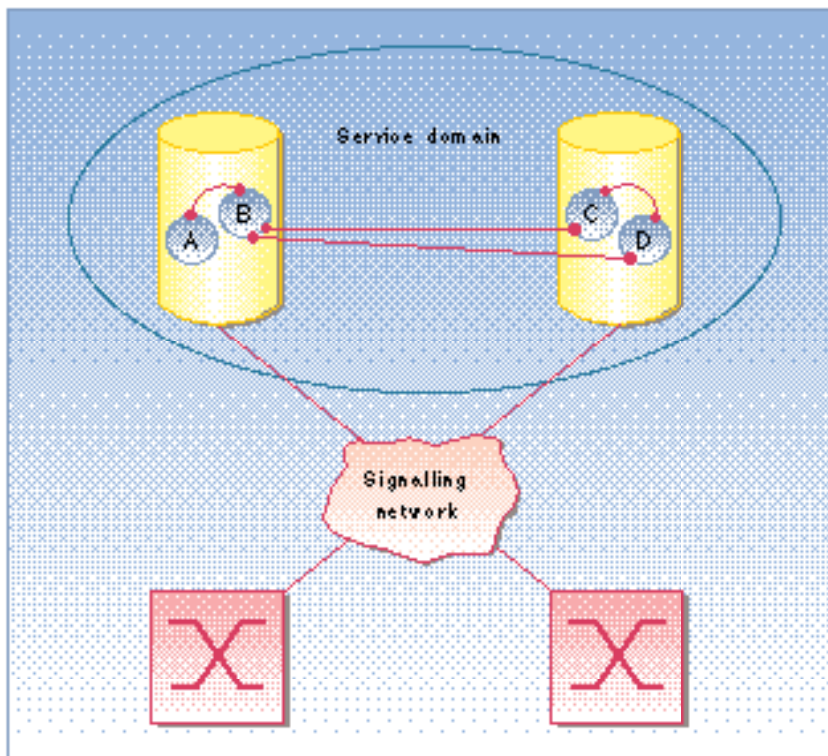
It may be convenient to let an attribute of one service object consist of other service objects. The call treatment attribute of the queue module in Figure 7 may thus contain conditional branching and announcement objects. Note also that the attributes required depend on the service: for the freephone example with routing on origin the conditional branching object only needs the origin attribute, while in order to provide routing conditioned on working hours it will require time of day, day of week and date attributes.

The attributes in one object are available for other objects in the service (or in other services interacting with it). Such interactions may involve reading attribute values, changing them, etc. Some attributes of an object may also be available for external run-time manipulation, for example by the service subscriber. In the example with the queue object it may be possible for the subscriber to change the value of the “number of server” attributes and to delete or insert new “identity of server” attributes. This may require that security objects are added to the service in order to avoid illegal manipulation on the attributes.

Our modular decomposition of services is so far incomplete since it has not included what is the basic element of most of them, namely the capability of providing and maintaining connections for information transfer between all parties involved in a call. This requires two important things:

- interaction between the services subscribed to and requested by each party
- interaction between the services and the physical network supporting the information transfer.

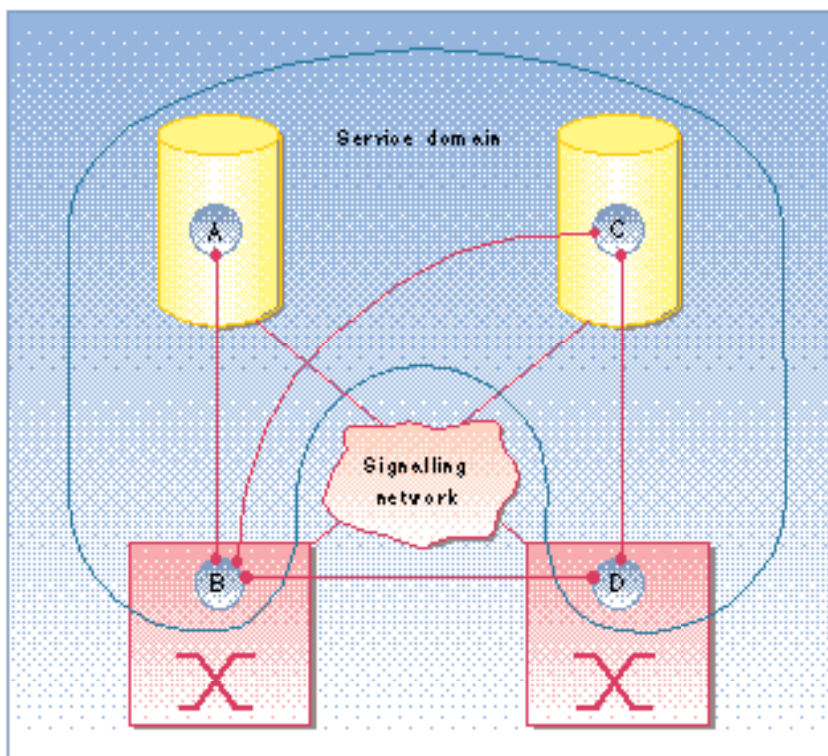
The purpose of the first is to sort out and resolve things such as



- incompatibility between services
- responsibilities of each service in executing the overall call
- ensuring that specific service conditions are not overruled (e.g. maintaining the integrity of closed user groups, respecting screening conditions, etc).

Studies of these problems have just begun. They will require investigation into areas of advanced computer science such as concurrency, distributed processing, distributed operating systems, dynamic rule based execution, etc. This is one of the biggest challenges in defining the evolutionary path of IN.

However, the existence of service interaction leads us to one of the basic conclusions of this section, namely the separation of service and physical connections. Before going into details, we must also look at how to handle the interaction between the service objects and the connections.



The first requirement concerning this interaction is one which we have already recognised as one of the objectives of IN, namely that a service should be executed in the same way on all types of network, i.e. the service execution must not depend on whether the physical network is ISDN, B-ISDN or a public data network. The second requirement is that the model must provide the service with a simple but consistent view of the network for each call. The simplest approach is to let the call configuration in the network be represented by a service object referred to as a virtual switch. This was also the approach taken by ETSI NA6 in their first definition of a call model (this model has later been abandoned by the CCITT and replaced by a much more network dependent and less intuitive model). The virtual switch object is illustrated in Figure 8. The object contains two types of attributes: leg and connection point where several instances of each may be present (legs and connection points may also be modelled as objects with their own set of attributes). The operations that can be done on legs and connection points then represent the manipulations that the service can do on the physical network. Note, however, that it is also necessary to represent other features of the network

Figure 10 Two ways of running the same service

*The service domain is distributed differently in the two cases but the resulting service is the same. Interaction between objects as indicated.*



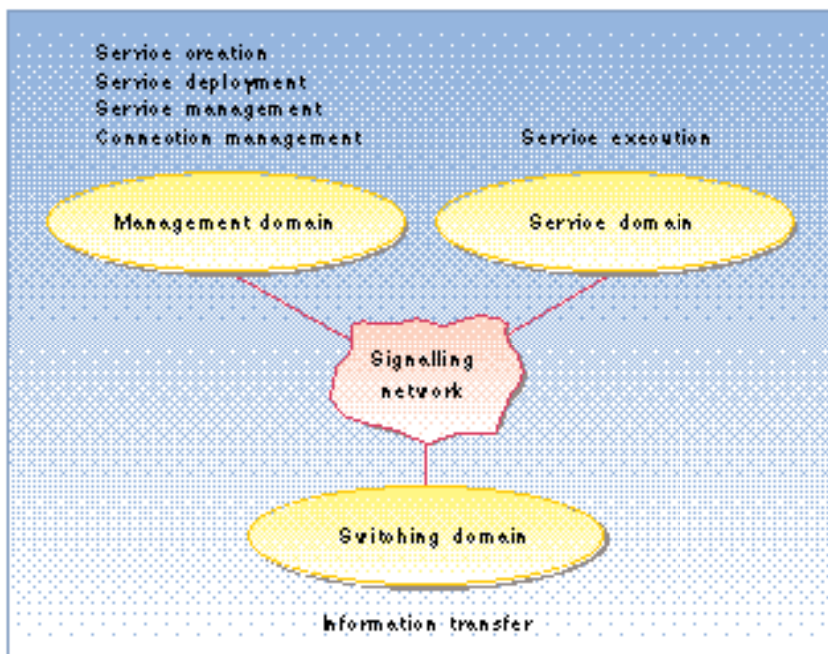


Figure 11 Network architecture including all three domains interconnected via a common signalling network. The architecture may consist of many instances of each type.

by objects available for service manipulation: traffic filters, various types of specialised resources, terminal devices, etc. These are not important for our presentation.

The virtual switch thus provides the service with a “need to know view” of the underlying network. In the hidden parts of the object we may conceal the protocols required for interacting with the network and any network dependent information. By carefully defining attribute values related to legs and connection points, one can make the part of the virtual switch object which is visible to other service objects independent of the actual network. For service execution it does not matter whether the connection point lies within one exchange or is distributed over several exchanges, or what switching technique is used: circuit, ATM, packet, etc. This makes the virtual switch also a reusable object.

At this stage we have at our disposal several service feature related objects and the virtual switch object. From these, several services can be built, each service exploiting the objects in different ways. Hence, by our approach we have reached another of the objectives of IN: reusability of service components.

Our approach has also led us to a new network architecture. By combining the service feature related objects and the virtual switch object, we can view the whole service execution process independently of the underlying physical network. This enables us to define two interacting domains: service domain and

switching domain as shown in Figure 9. The service domain contains all objects required for service execution while the switching domain represents connections and switches. As we saw for the tele-voting service, there are services where only the service domain is present.

In Figure 9 the signalling network (which already exists for signalling system No 7) has been included in order to clarify two points which have not been incorporated in present standards:

- The signalling network can provide for direct interactions between the users and the service domain. Hence, the triggering function (i.e. to identify where the service is to be processed) is the responsibility of the signalling network.
- The signalling network will provide the transmission medium for all inter-domain and intra-domain information flow, i.e. information flow between service objects.

### Flexible allocation of resources to physical entities

In the previous section we derived an architecture for IN from decomposing services into interacting objects. The decomposition was motivated by the reusability aspect of service software. However, this software must be run on machines.

The object oriented approach we have taken allows us more freedom in distributing the software. Figure 10 shows an example of a service consisting of four interacting objects distributed in two different ways. The resulting service is the same but the performance may be different: one distribution may be optimal with respect to processing delay, the other with respect to signalling network loading. From this it can be inferred that different strategies can be chosen for distributing the software and that the software may be distributed differently for different instances of a service or at different times, or when used on different types of networks.

Three prerequisites are required. First, the software must be portable, that is that it can be run on machines from different manufacturers. Second, there must be protocol support that can map arbitrary operations onto a general protocol stack. Third, all objects must be addressable within the signalling network. All of these represent challenges for future standardisation.

- It is not only software that can be distributed (and utilised) in an efficient manner in INs. There is an increasing need for customised network support such as mailboxes and customer specific announcements. Service interworking, protocol conversion, speech recognition and speech generation also require special resources in the network. It is a service feature and hence an IN task
- to determine that a special resource is required for the service
  - to find the appropriate resource and provide it with required information, and
  - to control its application in the execution of the service.

As for the virtual switch, these resources can be represented at the service level as objects and thus be included in a network independent manner.

Hence, IN opens for new ways of optimising the network. It should be noted that IN is likely to require large signalling resources. However, it will lead to better utilisation of other parts of the network.

## Management aspects

We have already seen that there is a close relationship between IN and TMN: management can be offered as a service to customers by use of IN features. However, there are three other areas where the merging is complete: service creation, service deployment and service management.

Service creation consists of designing services from service objects, assigning attribute values to them and defining rules for how the component objects should perform and interact in order to provide the required service. Service creation is also closely related to subscription management.

Service deployment concerns how, when and where service objects are to be loaded down into network equipment. This area opens for much optimisation. We have already seen that it may be possible to distribute the service in different ways in order to optimise its performance in one way or another. It may also be possible to deploy the service or parts of it at invocation or even at run-time.

The third aspect concerns management of the service itself. This may imply that an object may at the same time be defined as a managed object for TMN and a service constituent for IN, but where the visibility for the two applications may be different. An object which may have two usages is the statistics object: it may be used to record statistics on behalf of the service subscriber and it may be used by TMN for general network management purposes. However, queue objects, announcement objects, filters, etc. may also serve two purposes. This is one reason why a common object definition should be developed for IN and TMN.

TMN has been developed assuming only one domain, i.e. a domain where service and switching are integrated in the same machine. In a separated IN architecture TMN must supervise two domains where each domain will contain only elements of a call. Another point that complicates the picture is that IN will offer various kinds of mobility. One type of mobility we have already encountered, i.e. where the same service may be executed on different machines at different times. Another form of mobility is related to the

users where UPT (universal personal telecommunications) offers them the capability of making and receiving calls at arbitrary terminals. In such cases the service objects may be stationary while the access point varies. Of course, a combination of the two forms of mobility is also possible. In this respect IN represents a tremendous challenge to TMN.

Figure 11 shows an overall network architecture where all three types of domains have been included. In the figure it is indicated that several instances of the same domain may exist. For the switching domain this illustrates the existence of different types of network: ISDN, B-ISDN, packet data networks. For the service and management domains different development stages may coexist in the future.

## Conclusions

Based on its definition this paper has investigated the evolutionary potential of IN. The paper has not attempted to describe IN as it is now implemented and appearing in standards.

IN started about ten years ago in the USA and was aimed at solving some urgent problems with services such as freephone. Its core element was then and still is that some or all features of a service are executed on equipment external to the exchanges. Looking at the current status of IN the following conclusions may be drawn:

- The first real IN implementations are brought into operation now.
- These implementations are meeting very few of the objectives stated for IN.
- IN is coming at a time of great instability in telecommunications: ISDN is being introduced, many new supplementary services are being implemented before they have been fully specified, the market has been opened for competition of all kinds.
- IN standards are brought forward at a tremendous speed motivated by network providers to catch market shares and win in competition
- The IN standards are intended to solve short-term problems related to enhanc-

ing switching equipment for interacting with external equipment.

It took five years of standardisation work to get where we are today. It will probably take another ten years before we have exploited all possibilities that the definition of IN promises.

This paper has presented some of these possibilities focusing on which service capabilities IN can offer. It has also pointed at a possible evolutionary path that the development of network independent services may follow. However, the main problems are unsolved: service creation, service deployment and efficient use of network resources, service execution in a distributed environment, service management and merging of IN and TMN.

In order to solve them, new methodologies and a better understanding of what services are need to be developed. Some of these issues are treated in accompanying papers.



# The intelligent network service life cycle

BY HÅKON VESTLI AND RAYMOND NILSEN

## Abstract

*The intelligent network (IN) service life cycle is an abstract description of the structured, methodical development and modification process showing the main stages in producing and maintaining IN services. Starting with the life cycle for ordinary software, a life cycle model for IN services is developed. The description of this life cycle is supported by role models for each stage in the cycle. The role models are simple object oriented models that we synthesise into one composite model. We map this composite model onto a model of the physical entities that constitute a telecommunications network in order to exemplify how we can implement a service environment that supports the IN service life cycle.*

621.39.05  
681.324  
681.3.01

## 1 Introduction

In this paper we focus on the life cycle for services in a future intelligent network (IN). The study of the IN service life cycle will give us models of the environment in which IN services will exist. We intend to show by example how such models can be implemented. The terms *intelligent network*, *service* and *environment* need explanations in order for the reader to fully understand the focus of this paper. It is also necessary to explain some aspects of the modelling technique we will use. In sections 1.1 through 1.3 we will make some scene-setting remarks which will serve as brief introductions to each of the terms mentioned. In section 1.4 we discuss the particular methodology we use in the rest of the paper, and in section 1.5 we give an overview of the paper.

### 1.1 Intelligent networks

The common understanding of the term *intelligent network* refers to a telecommunications network that supports features such as fast service provisioning, vendor independence and efficient use of resources. It is currently expected that IN activities will give solutions on service creation, service interaction, pan-European services, modelling of specific services (such as Universal Personal Telecommunications and Virtual Private Networks), etc. This enhancement of requirements indicates that the major task to be performed during the development of an IN is to model all aspects of services including what they are, how they are specified, built, installed, activated, invoked, executed, etc. In this way we are able to uncover the detailed service requirements of a target IN architecture and finally identify a set of steps to evolve from today's networks towards the target architecture.

Most international activities on IN (in CCITT, ETSI, ECMA, etc.) seem to indicate that modelling of services in an IN primarily should specify service func-

tionality. In (1) we follow the idea of complete separation of service related functionality and switching related functionality on the design level. In this context separation implies separate modelling of service related and switching related functionality, bridging the two models with a carefully designed interface. An important consequence of this separation is that it conveniently supports the strive towards network independent services. The aspect of separate modelling has a built-in notion of independence in the sense that when we are modelling one part, we can only see the other part from our side of the interface, i.e. when we are modelling service related functionality, we have no knowledge of switching related functionality except for those functions offered over the interface. See Figure 1.

The proposal allows general service logic to be considered without reference to the actual network supporting the services thereby achieving network independent service specifications. This separation simplifies the problem of modelling services because it allows us to focus on general properties of services rather than switching dependent issues. In fact, the

separation represents a practical way of defining subproblems and it allows for easier incorporation of results from work done outside the telecommunications community.

### 1.2 Services in an intelligent network

In principle services in an intelligent network can be thought of as applications in a huge distributed computing system. It seems to be fruitful to utilise this analogy as it allows us to reuse much of the results from areas such as software analysis and design, programming techniques, formal description techniques, etc. It provides us with a flexible notion of the concept of a service which simplifies development of methods for fast, secure, and efficient manufacturing of service software.

In this article we will adopt a view of services according to the 'black box' principle, i.e. the input and output of the service are the only observable features of the service. In other words, we do not say much about the internal structure and functionality of them - the focus is on the environment they exist in.

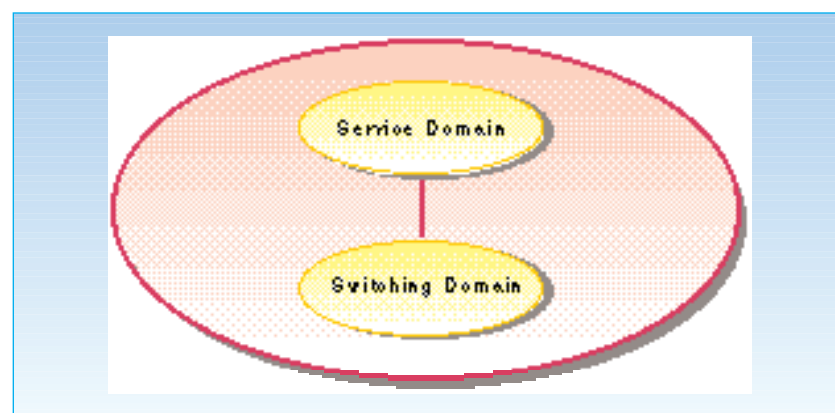


Figure 1 The intelligent network

*In an intelligent network functions in the service domain use functions offered from the switching domain. The idea of separation is built upon the client-server principle, that is, the service domain is a client of the switching domain, which is a server of connectivity in the telecommunications network.*

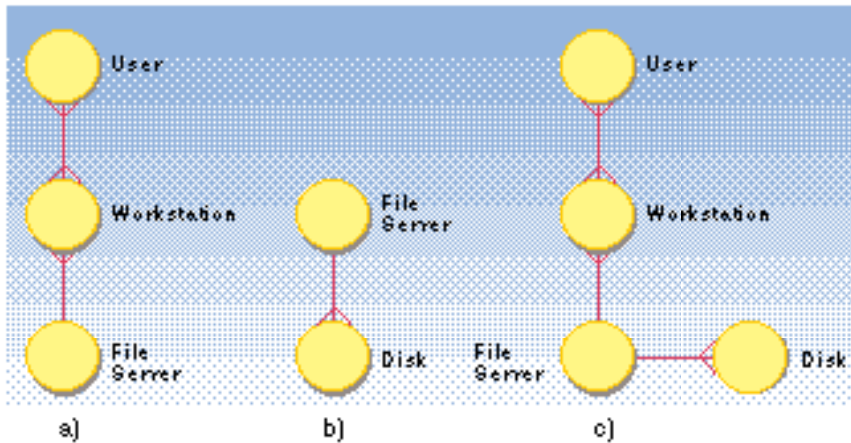


Figure 2 Role modelling

In Figure 2a) we model that a user works on a workstation which is connected to a central file server. That the workstation uses a file server is transparent to the user. In Figure 2b) we model that the file server controls a set of disks. We see that on the one hand, the file server has to serve workstations, while on the other hand, it has to control the disks in order to be an efficient file server. These two File Server roles are therefore not identical. In Figure 2c) we have made an object model based on the synthesis of the role models in Figures 2a) and 2b).

Circles in Figures 2a) and 2b) represent roles, while the circles in 2c) represent objects. Lines between roles or between objects indicate that messages flow between them. Crow feet indicate cardinality. We have for example that a user can use many workstations and a workstation can serve many users (a many-to-many relationship). A workstation is related to one file server while the file server serves many workstations (a one-to-many relationship). We have a one-to-one cardinality when a line is without any crow feet. Note that if a role has a one-to-many relationship to another role, this means that the former can be related to one, two, or many of the latter. For many-to-many relationships this comment applies in both directions.

### 1.3 Service environment

We define a service environment as a model of the surrounding functionality that is necessary for a service to proceed through its life cycle. The life cycle of a service will be discussed in detail in section 2. The set of models resulting from the discussion in section 2 will be used as a basis for the specification of a total runtime environment for services. The scope of this paper is to realise some guidelines for how a service runtime environment must behave.

There is no sharp boundary between what can be defined as a service and what can be defined as the service environment. We can, however, introduce one general principle for how we would like to separate between services and the service environment, namely that features that are unique to a service should be defined to be within that service, while features that are common to many services should be defined as parts of the environment. It is also of importance to design the service environment as general as possible in order to be able to support future services.

### 1.4 Modelling technique

To support the descriptions of the various life cycle stages, we use parts of an object oriented modelling technique called Object Oriented Role Analysis, Synthesis and Structuring (OORASS), see (4, 5) where the following description primarily is taken from:

In general, an object oriented system may be thought of as a kind of perfect bureaucracy, where each object is like a clerk who is responsible for a certain part of the whole operation (the clerk plays a role in the undertaking of the operation), and where things happen when the clerks pass messages to each other. Object oriented design is then to design the pattern of this interaction and to delegate responsibility to the individual objects in such a way that the total system of objects is as simple as possible while it shows the desired behaviour.

OORASS is a comprehensive method for industrial analysis, design, implementation and configuration of object oriented systems. The OORASS methodology comprises Role Modelling, Object Speci-

fication, Class Implementation, Structure Configuration and System Instantiation. In this paper we will only use Role Modelling, which is separated into role analysis and role synthesis.

In this paper, we use role analysis to study the IN service life cycle. Each life cycle stage will be supported by a role model that offers an implementation-independent description of interacting roles. A role is to be understood in a traditional sense as an entity's task or duty in an undertaking - the focus is on behaviour. For each role in a role model we give it a name, describe its responsibility, determine which other roles it needs to know about, and determine the messages it sends to these. If the behaviour of a role is complex, it can be represented by its own role model on a lower level.

When we subdivide the area of concern (the IN service life cycle) into smaller areas (stages), creating role models for each subarea, we reduce the modelling problem to manageable proportions, but create the new problem of integrating the smaller models into a model of the entire area of concern. In OORASS, the solution to the integration problem is called role synthesis: Given a number of role models, we create a set of objects, each of which may play roles from different role models.

For an example of role modelling, see Figure 2. We have simplified and slightly modified the OORASS way of drawing role models in order to support our particular needs.

### 1.5 Paper overview

In section 2 we introduce and discuss our definition of the IN service life cycle. Starting with a description of the life cycle for ordinary software, the IN service life cycle is developed. We describe in detail each stage and support these descriptions with simple role models. In section 3, these role models are synthesised in order to get a composite model. This model is then mapped onto a model of the physical resources that constitute a telecommunications network, in order to exemplify how we can span the gap between high-level modelling to concrete implementation, yielding a service environment that supports the IN service life cycle. In section 4, we give our concluding remarks.

## 2 The life cycle model

We regard services as software applications to be run in the telecommunications network. Since services are software we can introduce the discussion of the service life cycle by skimming through the life cycle for ordinary software systems.

By common usage a life cycle model is an abstract description of the structured, methodical development and modification process typically showing the main stages in producing and maintaining executable software (3). A typical life cycle model is the following:

- 1 Analysis: Analysing the user's requirements.
- 2 Specification: Describing what the system should do, as seen from the user.
- 3 Design: Describing the system's interfaces, functionality and structure as the designers intend to implement them.
- 4 Implementation: Producing the programs.
- 5 Installation: Installing the programs on the target computer system.
- 6 Use and maintenance: As the software is used, errors and new requirements are usually uncovered. Maintenance is the task of modifying the software so that the errors are eliminated and the new requirements are taken care of.

Iteration between the phases is often necessary, but it should preferably be confined to successive steps.

What is specific about intelligent network services software? First, the target computer system on which the software is to be installed, is a huge distributed telecommunications network. Second, there is a large amount of users, subscribers and of potential services. Third, there are severe constraints regarding response time and availability of the services in the network. These factors are important in the modification of the ordinary software life cycle so that it becomes a service software life cycle. The service life cycle we suggest is shown in Figure 3.

### 2.1 The actors in the IN service life cycle

In discussing the actors we will also mention the various roles the actors will be represented by in the role models to follow in sections 2.2 through 2.4.

#### *The User:*

The User is the party that wants to use services. That is, the Users constitute the target group for a service. The subscriber himself can be the only user, alternatively the subscriber intends the service to be used by other parties. The User will be represented by the Service Requester role.

#### *The Subscriber:*

The Subscriber wants to subscribe to a service. This may require the creation of the service if existing services cannot be used to fulfill the Subscriber's requirements. The Subscriber is represented by the Subscriber role and indirectly by the Service role.

#### *The Service Creator (also called Service Provider):*

The Service Creator is a party that has a license for producing telecommunications service software to be installed in the telecommunications network. The Service Creator receives requests for creating a service from a subscriber or can initiate the creation of services himself, in which case the Service Creator also acts as Subscriber. General services can be made intended for being subscribed to by many people. Services can also be tailor-made to one particular subscriber. The service creator can be the network operator, a network equipment vendor or in general an independent software manufacturer. The Service Creator is represented by the Service Creator role, which will need various Handler roles: for Service Descriptions, Service Specifications, Service Designs and Service Implementations.

#### *The Network Operator (also called Network Provider):*

The Network Operator is the party that controls the physical resources required for offering a telecommunications network for use by the parties mentioned above. The Network Operator will therefore be represented by a multitude of roles: the Database, the Database Handler, the Installation Manager, the Activation Manager, the Allocator role, the Service Execution Manager, the Switching Domain role and also the roles con-

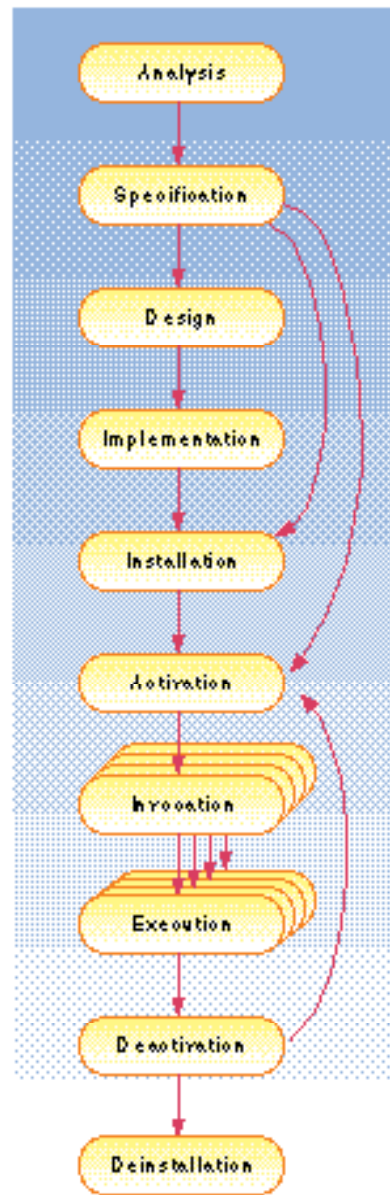


Figure 3 The life cycle model for services

The figure describes the various processes a service goes through, from the analysis of the subscriber's requirements to the service to the deinstallation of it. After analysis, the service is specified, designed, implemented and installed. After activation the service can be invoked an arbitrary number of times (including none), each invocation to be followed by the execution of the service. At some time, the service is deactivated, and eventually deinstalled. Arrows between two processes depict that the service can go from the first process to the second.



cerning Communications, Monitoring, Resource Management, Traffic Management and Data Management.

These actors constitute a hierarchy that can be described by client-server relationships:

- The User is a client of the Subscriber, which is a server of services.
- The Subscriber is a client of the Service Creator, which is a server of service software.
- The Service Creator is a client of the Network Operator, which is a server of the physical telecommunications network.

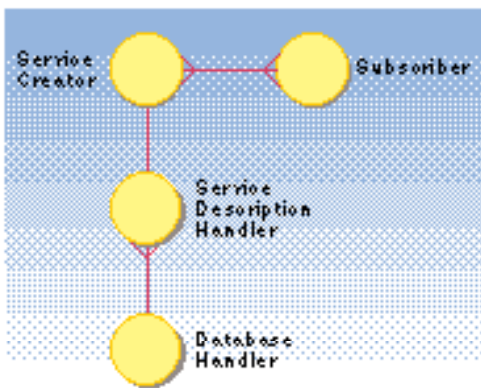


Figure 4 The Service Analysis role model

The Subscriber negotiates with the Service Creator in order to develop the Service Description, which is handled by the Service Description Handler. The description is stored by the Database Handler.

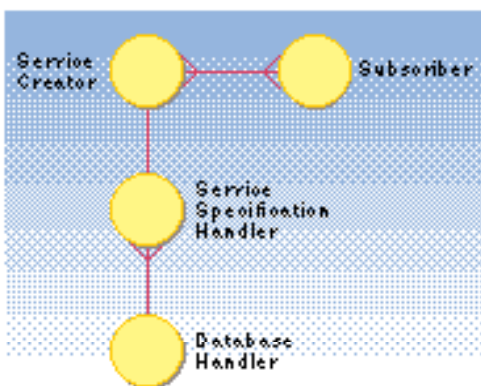


Figure 5 The Service Specification role model

The Service Creator develops a Service Specification, with additional support from the Subscriber. The specification is handled by the Service Specification Handler and stored by the Database Handler.

## 2.2 Early phases

The early phases of the service life cycle comprise the activities from the service idea is conceived until the service is activated and ready to be invoked, i.e. the analysis, specification, design, implementation, installation, and activation phases.

### Phase 1. Analysis of the subscriber's requirements

The main task is here to produce a definition of the characteristics of the service, focusing on how the subscriber wants the users to see it. This implies close cooperation between the service creator and the subscriber in order to bridge the gap between the subscriber's requirements and the service creator's assessment of what is a feasible service. See Figure 4 for the Service Analysis role model.

### Phase 2. Service specification

In this phase, the service is specified according to the subscriber's requirements. The output from the previous phase must determine this specification as unambiguously as possible. The specification should be written in a formal language so that it can be checked for consistency and also so that programs generated later can be verified against it. Another reason for formalising the specification is that it increases the possibility of automation in the design and implementation of the service.

The specification is supposed to be developed under the constraints or restrictions the network as a whole imposes on all services. That is, we cannot in practice allow for arbitrary applications to be installed in the telecommunications network - there has to be a strict control of which applications are allowed as services. Therefore, it must be checked that the specification represents a meaningful telecommunications service, and that the service does not compromise the network, the network operator or other network users.

It is important that the subscriber can confirm that the specification produced satisfies his requirements, i.e. that it represents the service he wants. This is usually called validation. One way to validate a service is to rephrase the specification in a language the subscriber is accustomed to, e.g. natural language.

During this phase, the service creator will find out to what extent existing services can be used. It may be that the service is unprecedented in the network, which means it must be specified, designed, implemented and installed. On the other hand, it may be that the subscriber's requirements are fulfilled by subscribing to an existing service, and the design and implementation phases can therefore be omitted. If that existing service is adequately installed to serve the new subscriber, the service is ready for activation of the new subscription, but if not, then the service goes to the installation phase first.

See Figure 5 for the Service Specification role model.

### Phase 3. Design

To facilitate rapid and efficient introduction of new services, we envisage extensive reuse of specifications and software. In the design phase services are designed by assembling reusable software components in a controlled manner. Most of the software components are already designed in detail - the design of the service concerns the problem of putting various software components together according to the specification. This should be a trivial task, supposing there is a well-defined mapping between the specification constructs and the design objects.

We can envisage that such software components are standardised, in order to facilitate international IN services. On the other hand, a service creator must be allowed to produce software not covered by the set of standardised software components, both to support the subscribers' particular needs and also to get a competitive edge in the service market. This means that new software components can be made at the service creator's discretion. This new software has to be designed in detail in this phase, so that the service as a whole is completely designed.

See Figure 6 for the Service Design role model.

### Phase 4. Implementation

The implementation phase concerns the production of complete programs that

satisfy the specification of the service. The implementation will ordinarily contain both new software and reused software. Reused software is taken from a software library while the new software has to be implemented now.

When we have the complete source code, we may be able to formally verify that this code satisfies the specification. There can be features required by the subscriber that are not formalisable. Such requirements will lie outside the scope of verification, and to complement the verification activity, testing or simulation of the service should be performed.

See Figure 7 for the Service Implementation role model.

### Phase 5. Installation

Performance is of utmost importance in the telecommunications network. This concerns the speed with which services are executed and the requirement that all services are available at all times. To cope with this requirement, we need a decentralised solution, where software and data can be distributed in the network instead of residing in a centralised network node. Distribution concerns both how one unit is divided into different parts and how these parts are replicated in the network. Note that when software or data are replicated we get problems concerning the consistency between different copies of the same unit.

The distribution of service software is a typical optimisation problem, where the objective is to minimise the volume of data and the number of messages transmitted across the network during the execution of services. Parameters in this optimisation problem are the locations of users, the frequency of usage, the network topology, database sizes, etc. Ideally, once the service is installed it should be possible to dynamically modify the distribution of software and data as these parameters change.

When the distribution of the software is determined, the source code can be compiled into machine code depending on the hardware of the network, and then this can be installed.

We should have seamless addition of services. That is, the actual installation

should be performed without any interruption of the network.

See Figure 8 for the Service Installation role model.

### Phase 6. Activation

Activation means that the service is made available to the users, that is the service can now be invoked. We refer to the activation of a service in three different cases:

- The service is freshly installed, and has not been in use.
- The service has been used by other subscribers, and this new subscription is handled by modifying the new subscriber's service profile.
- A subscriber modifies a service's parameters by operating directly on his service profile (not all subscribers have the right to do this). The modification is effectuated by activating his modified service. This can be viewed as a maintenance activity. Security requirements must be addressed at this point.

Activation can be limited to a certain set of users, a certain geographical area, etc.

See Figure 9 for the Service Activation role model.

## 2.3 Invocation and execution

The service is now activated and ready to be invoked. Here we describe in more detail the invocation and execution phases.

### Phase 7. Invocation

A service is invoked when a user sends a request for the execution of the service to the network. The network has to identify the user requesting the service, the service that is requested (possibly demanding user authentication or authorisation), retrieve the information necessary to start the service (including data or program code), and finally initialise it. See Figure 10.

### Phase 8. Execution

The service has now been invoked and initialised and is now executing. A role model to support the description is shown in Figure 11.

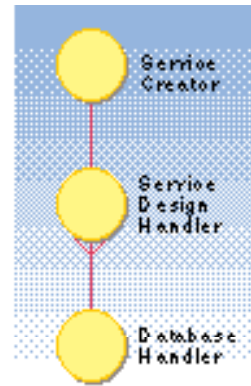


Figure 6 The Service Design role model

The Service Creator develops a Service Design, which is handled by the Service Design Handler. The design is stored by the Database Handler.

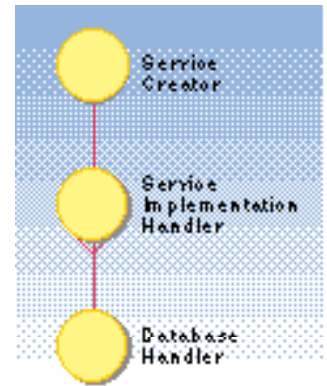


Figure 7 The Service Implementation role model

The Service Creator develops a Service Implementation, which is handled by the Service Implementation Handler. The implementation is stored by the Database handler.

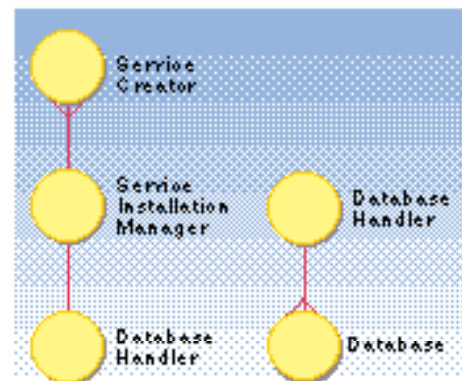


Figure 8 The Service Installation role model

The Service Creator now sets parameters for the installation and requests the Service Installation Manager to install the service. The Database Handler is responsible for the actual storage throughout the network. That is, the Database Handler controls the various Databases throughout the network.

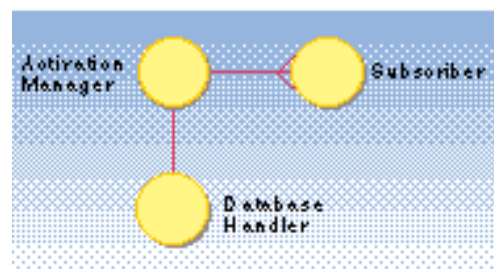


Figure 9 The Service Activation role model

The Subscriber requests the Activation Manager to activate his service.

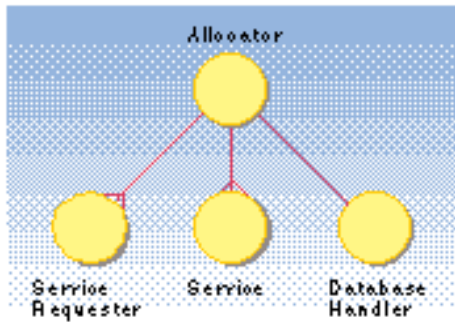


Figure 10 The Service Invocation role model

The Service Requester requests the initiation of a service. The Service Requester can represent a human user of the network (in reality his terminal agent), it can be a computer application such as another service, etc. The Allocator accepts requests for the execution of services, and processes these. The Service is a computer application and has a functionality of arbitrary complexity.

*The Service as a Service Requester*

The service to be executed can be a simple service, e.g. a two-party-call. On the other hand, the service can be a service application that on the basis of a dialogue with the end user determines another service that is to be invoked. For example, imagine that the end user has a computer as his terminal equipment. Imagine further that the user is running a window manager on his computer. One of the menu items the user can choose in his root window can be a telecommunication service called, say, TeleMenu.

When TeleMenu is invoked, it presents the end user with a menu containing a list of all his telecommunication services, e.g. video conference, remote database access, remote interactive video. When choosing one of these, the service TeleMenu then acts as a service requester,

requesting the invocation of the chosen service. The point is that executing services can invoke other services. This is a part of the problem area of service interaction.

*The Service's Interface to the Switching Domain*

Since it is desirable to define services independently of the type of network, the running service software should not have direct access to the interface between the service domain and the switching domain. The service applications should rather use roles in the service domain that know how to talk to applications in the switching domain, i.e. roles in the service domain representing switching functionality. For this purpose a Communications role can be constructed that represents the switching domain's functionality regarding connections in the telecommunications network. For instance, if the service application wants to establish a connection between two parties, it requests the Communications role to do this. The Communications role maps this request onto the protocol between the service domain and the switch-based connection control functions.

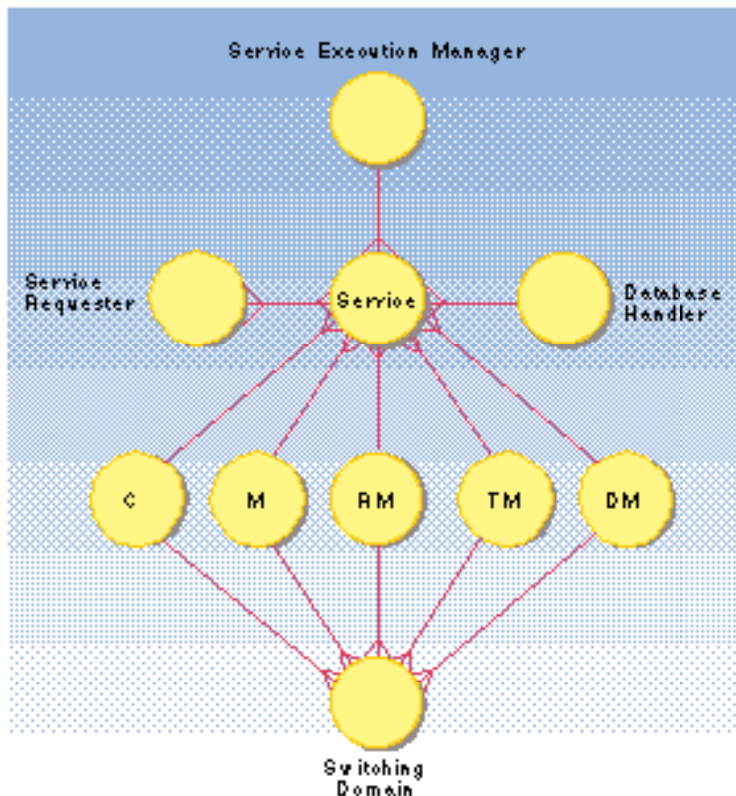


Figure 11 The Execution role model

The Service Execution Manager monitors and controls running services. It can also receive requests from Services regarding status of the network or other information (e.g. time and date). The Switching Domain represents the switched network, that is, the physical telecommunications network. The knowledge of what kind of network this actually is should be of less relevance to the Service, which controls the Switching Domain via the intermediary roles called Communications (C), Monitoring (M), Resource Management (RM), Traffic Management (TM) and Data Management (DM).

Extending this example, a set of roles that together with the Communications role cover the switching domain functionality can be constructed. The Monitoring role offers monitoring of connectivity in the telecommunications network. The Resource Management role offers management of resources in the telecommunications network. The Traffic Management role offers management of telecommunications network traffic. The Data Management role offers management of data in the switching domain. Note that these intermediary roles can be addressed by several services. This allows for one service transferring the control over for example a Communications role to another running service. The design of these roles is inspired by ETSI's Connection Control Model (2).

**2.4 The last phases**

Sooner or later, the service is getting obsolete. This requires deactivation and deinstallation.



## Phase 9. Deactivation

Deactivating a service is making it unavailable to users, that is, it can now not be invoked. The service can be reactivated at a later stage. Deactivation can be performed for a certain set of users, a certain geographical area, etc. We can here use the same role model as for Service Activation, see Figure 9.

## Phase 10. Deinstallation

Deinstallation occurs when the subscriber wants to quit his subscription, or if the subscriber for some reason or other is not allowed to subscribe to this service. Reusable parts of the specification and program code can be retained, but the service software should be deleted from active network resources unless it is shared by other subscribers. We can here use the same role model as for Service Installation, see Figure 8.

## 2.5 A scenario

Let us give an example of how we think this life cycle would look like in a concrete case. This is a very simplified example, and it is only meant to illustrate the previous discussion of the service life cycle.

Gary, the boss of a travel agency, has plans for opening a new chain of travel agencies across the country. He is already heavily using IN services, and has now an idea for a tailor-made IN service to be used by his customers in his new travel agency chain. Let us proceed through the IN service life cycle to see how his idea for an IN service is gradually refined into a new service.

### 1. Analysis of the subscriber's requirements

Gary makes an appointment with the service consultant Berke at the closest authorised IN service creator, a company called IN Services Ltd. When they meet, Gary comes up with his service idea. His travel agency chain has offices in four cities. Gary wants one phone number to cover the entire chain, he wants his customers to be able to call his offices free of charge, and that the calls are routed to the nearest office. If all extensions at one office are busy or the office is closed at that moment, the caller will be rerouted to another office. When there is a call

outside working hours, the caller is to be told when the office opens. Gary and Berke discuss this idea thoroughly, uncover all details Gary has not thought of on his own, and they end up with a document written in natural language that describes the service. They meet a couple of days later to make the last adjustments to this description.

### 2. Service specification

Berke now takes the service description with him to his workstation, where his specification tool is running. He finds no existing specification he can reuse, so he has to go through the service description very thoroughly and transcribe it into a formal language. The specification he produces looks a lot like mathematics, but is in fact rather easy to read. In the specification tool, there are service checkers that go through the specification to find inconsistencies and things that could disrupt the network. After a couple of iterations, Berke is satisfied with the specification. He puts it through a translator, gets a natural language version of the formal specification, and makes an appointment with Gary. At that meeting, Berke goes through both the formal specification and the rephrased version with Gary to get a confirmation that this is really the service Gary wants. It turns out that Gary is very happy with the specification.

### 3. Design

Berke now uses his service design tool to transform the specification into a concrete design of the service. This is a rather trivial job, since there is a well-defined mapping between the formal specification language and the object oriented language the design is written in. There are some small enhancements to the design, but Berke is satisfied with the design after a relatively short period of time.

### 4. Implementation

The source code of the software components that were reused in the design, is retrieved from a software library. Berke only has to implement those small enhancements he made to the design in order to get a complete implementation. The resulting piece of software has a well-defined interface to the IN service environment, and is supposed to fit in

without problems. Berke is convinced of that after he has churned his code through a verifier to check it against the specification.

### 5. Installation

The appropriate parts of the software produced are compiled and put in IN nodes in the four cities mentioned above. System software on each node is modified to take account of the new service. Central registers are updated with Gary's new subscription, charging rates are stated, etc.

### 6. Activation

When all the software is properly installed and all registers are updated, the service can be activated. That means that the service can now be used. At activation time, Gary is extensively running commercials in order to make the market aware of his new travel agency chain and of the IN service the customers can use in order to reach its offices.

### 7. Invocation

Paulina has for a long time been thinking of taking a well-deserved vacation to the Seychelles. But the trip is expensive, and she has not had the time to do it. Now she sees in the paper the commercial for Gary's travel agency chain, which includes a good offer for trips to the Seychelles. She sees the freephone number and runs for the phone.

The software in Paulina's terminal that transfers the digits to the network is playing the Service Requester role. The Allocator role in the network analyses the digits, loads the data concerning the dialled number from the Database Handler role and 'understands' that Paulina is requesting Gary's freephone service. The Allocator then puts the service's machine code in memory, and initialises the service to be run.

### 8. Execution

The Service role analyses Paulina's number and uses the Communications role to set up a connection to the nearest office, which the service supervises by using the Monitoring role. The Service Execution Manager role supervises the running of the service. Paulina gets the connection and ends up with ordering one of the

low-price trips to the Seychelles. As she breaks the connection, the Service role sends messages to the Database Handler in order to update Paulina's and the travel agency's billing information, and also some network statistics, and is then finished.

### 9. Deactivation

After a lot of invocations and executions of the service all over the country, and

during a period of a couple of years, Gary finds out that the service is too expensive for his travel agency chain. He calls Berke and asks him to deactivate his service, so that customers can no longer use the freephone number. Another of Gary's services is modified in order to give the travel agency chain the same phone number as the rest of Gary's travel agencies, so that people can still reach them. However, he wants to have the

chance to activate the service later, if the customers' responses should turn out to be too negative.

### 10. Deinstallation

Some time passes by. Gary's customers have not been too negative to the change, so Gary makes the decision to deinstall the service. He makes a call to Berke, who takes care of the deinstallation.

## 3 Realisation

Having discussed the stages of the life cycle and presented simple models for describing some of their most important characteristics, we will show how these results can be used to derive an implementation of a system that supports such a life cycle. We will show the flexibility of the results with respect to allocating functionality to physical entities.

The procedure for specifying the implementation goes through two steps. The first step is to transform all models in section 2 into one big model in terms of objects and relations, as described in section 3.1. In the second step we map objects onto physical entities thereby finalising the specification necessary to do the implementation of the system. This is described in section 3.2.

### 3.1 The synthesis of models

In section 2 we have presented one role model for each stage in the life cycle. Although many of these models are still lacking in detail they remain realistic and suitable for further elaboration and development. In this section our aim is to create one model described in terms of objects and their relations from a set of models described in terms of roles and relations between roles. The principles for doing this synthesis have been explained briefly in section 1.4.

Before proceeding with the synthesis we give a short summary of the modelling results of section 2 as we for each role model list the names of the roles that belong to it:

- Service Analysis role model: Subscriber (1), Service Creator (2), Service Description Handler (3) and Database Handler (4)

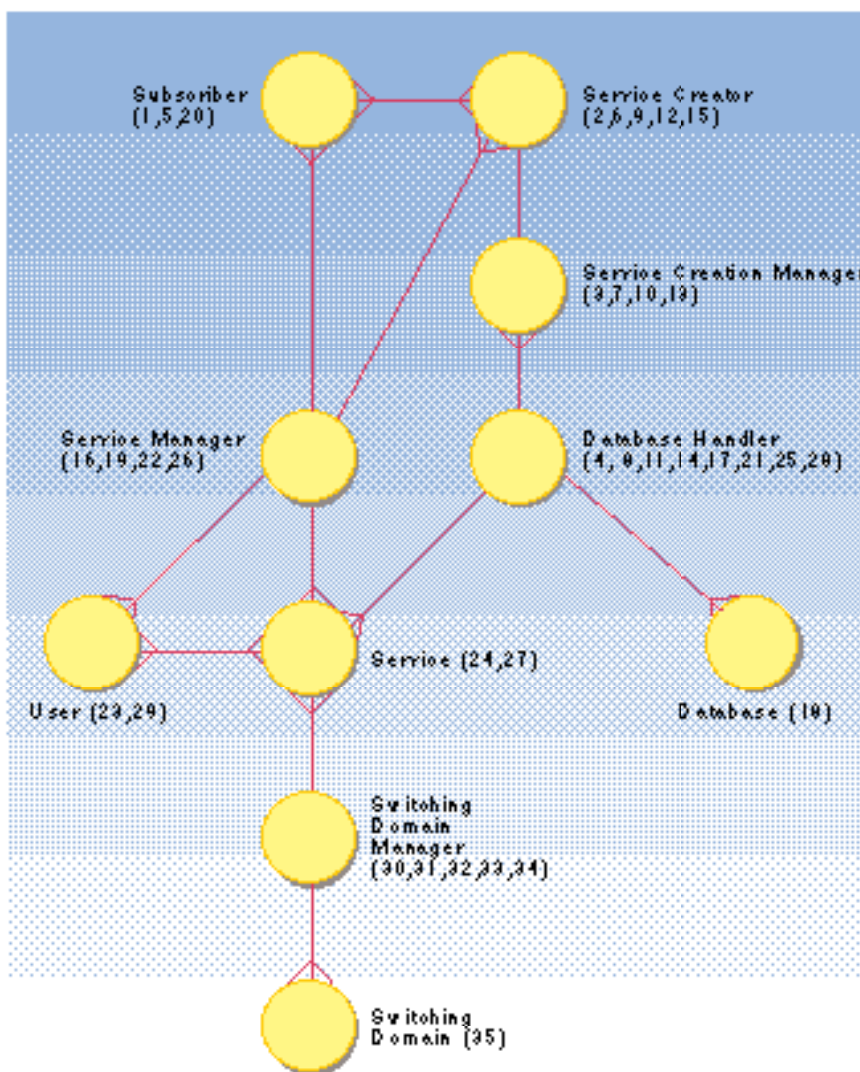


Figure 12 A synthesised object model

Here we show a composite model based on the role models in section 2. The process of synthesising role models is far from trivial, and a whole range of alternative object models can be made. For example, if we would like a stronger separation between the behaviour of the Service Creator in the Analysis and Specification stages and the behaviour of the Service Creator in the Design, Implementation and Installation stages, we could replace the Service Creator object by an Analysis-Expert object playing roles 2 and 6 and a Design-Expert object playing the roles 9, 12 and 15.

- Service Specification role model: Subscriber (5), Service Creator (6), Service Specification Handler (7) and Database Handler (8)
- Service Design role model: Service Creator (9), Service Design Handler (10) and Database Handler (11)
- Service Implementation role model: Service Creator (12), Service Implementation Handler (13) and Database Handler (14)
- Service Installation role model: Service Creator (15), Service Installation Manager (16), Database Handler (17) and Database (18)
- Service Activation role model: Activation Manager (19), Subscriber (20) and Database Handler (21)
- Service Invocation role model: Allocator (22), Service Requester (23), Service (24) and Database Handler (25)
- Service Execution role model: Service Execution Manager (26), Service (27), Database Handler (28), Service Requester (29), Communication Manager (30), Monitoring Manager (31), Resource Manager (32), Traffic Manager (33), Data Manager (34) and Switching Domain (35).

Note that although many roles have the same name in different role models they are not identical. For example, the subscriber role may have a very different behaviour in the analysis phase compared to the specification phase. However, the detailed modelling of behaviour is outside the scope of this paper. Also note

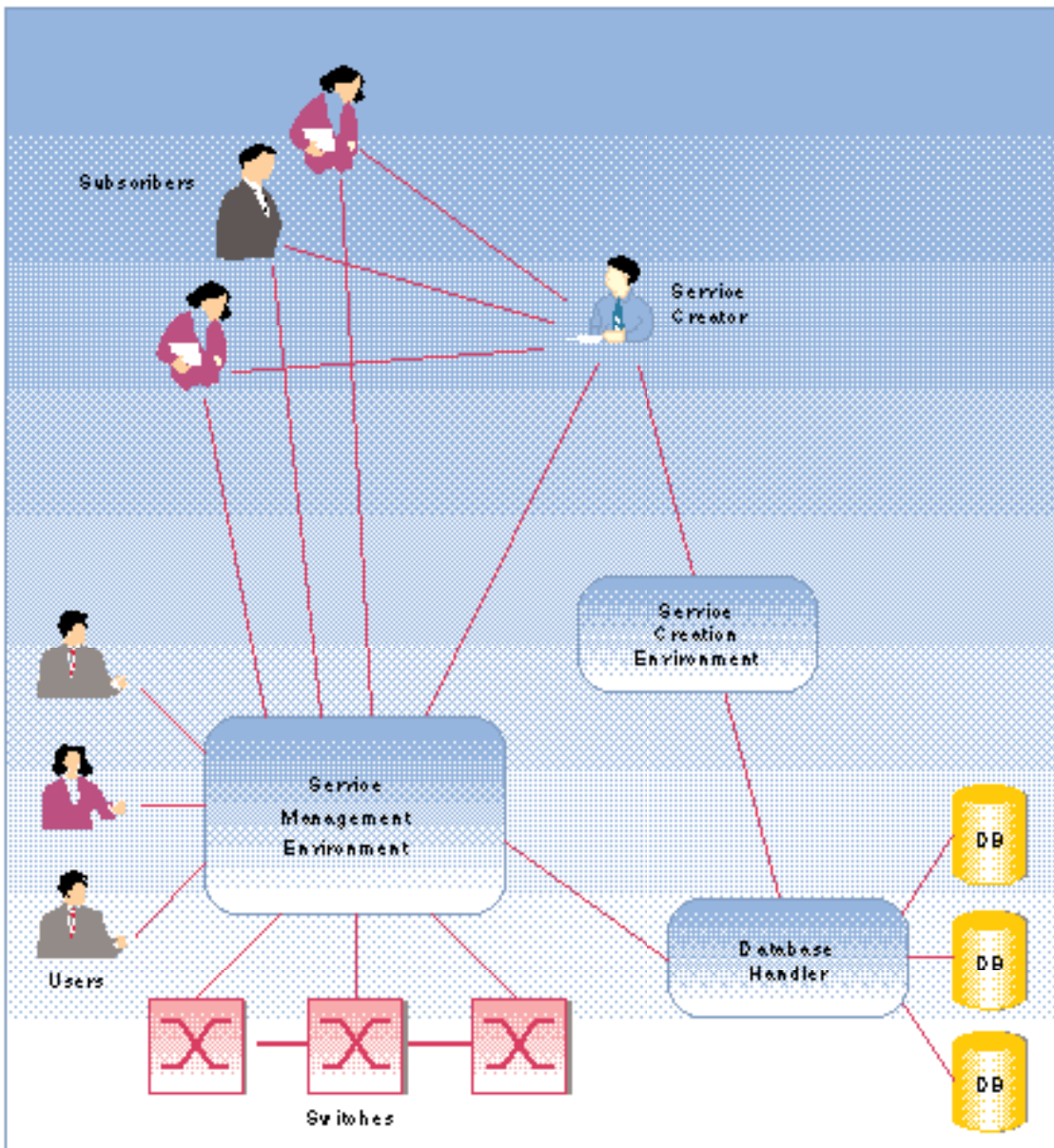


Figure 13 An intelligent network architecture

This architecture could be extended to explicitly handle the distribution of functionality. This can be done by further detailing existing objects to include distribution related behaviour, e.g. by adding role models on lower levels in section 2.



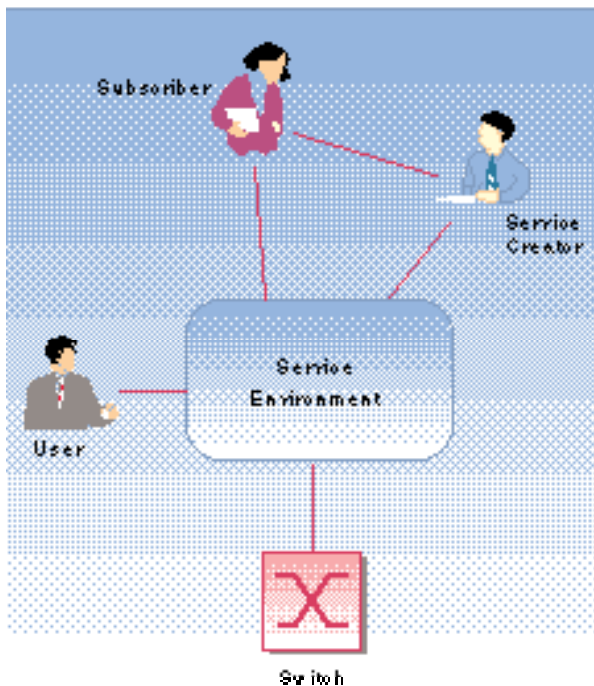


Figure 14 An intelligent network laboratory architecture  
The introduction of a simulator program to compensate for the removal of the switch would simplify the implementation and reduce the cost for a laboratory.

that for the sake of convenience we will use the numbers attached to the roles as references in the text rather than their names.

We now identify the objects of our object model and allocate to them the roles they will play in the system. Effectively, we constrain our models by defining which roles must be played by the same object. The message flows between objects are determined by the allocation of roles to objects. If two roles in a role model, being connected by a message flow, are allocated to two different objects in an object model then there will be a message flow between the objects that corresponds to the message flow in the role model. Similarly, synthesis maintains the cardinality of relations between roles. The total message flow between two objects in an object model defines the interface between them.

Let us define an object called Subscriber and have it play roles number 1 and 5. Similarly, let us define an object called

Service Creation Manager and have it play roles number 3, 7, 10 and 13. We can continue like this until we have allocated all 35 roles in the list to some object. Our object model is shown in Figure 12.

The synthesised model constitutes a specification for the implementation of the behaviour captured in the object model. Note that such a specification is independent of programming language and does not yet specify any distribution of objects onto physical entities. This distribution will be described in the following section.

### 3.2 Mapping objects to physical entities

This section is concerned with the allocation of objects of an object model onto a set of physical entities. When we talk about physical entities in this context we mean either human beings or some sort of machine capable of processing software. In the case where objects are allocated to human beings, their expected behaviour is specified by the roles that make out the particular objects and they can only interact with other entities (obeying cardinality information) using the messages that are defined between the objects they represent.

On the basis of the object model we are now in a position to create many scenarios for how to map behaviour to physical entities without being unduly constrained by predisposed (and often unnecessary) ad hoc decisions. It allows us to reuse our modelling results in highly differing alternative physical architectures. Figure 13 presents a scenario that is similar to descriptions of realistic initial IN implementations whereas Figure 14 shows how the same behaviour can be introduced into a physical architecture for experimental use like simulations or demonstrations. Excluding the Switches, these implementations will belong to the service domain described in section 1.1 thereby adhering to the principles for achieving network independence.

In Figure 13 we continue the example from section 3.1. We suggest to allocate the Subscriber, User and Service Creator objects to human beings. Note that the object model allows several Subscribers, Users and Service Creators. We choose to allocate the Service Creation Manager

to a computer which will be labelled the Service Creation Environment. This machine can be a desktop office machine in the Service Creator's office just as described for Berke in the scenario in section 2.5. A database machine will be responsible for all aspects of data access which in our models include the Database Handler object and the Database object. The Switching Domain object is a representation of all switching and transmission functionality in the telecommunication network including necessary interface functionality to the Switching Domain Manager object and may be assumed to be implemented by modern switching machines and transmission equipment between them. The service control and execution can be done by a separate machine labelled the service management environment and has allocated the Service Manager, Service and the Switching Domain Manager objects.

In Figure 14 we depict an architecture that could be suitable for an IN laboratory activity. The Service Creation Manager, Database Handler, Database, Service Manager, Service and Switching Domain Manager objects have been allocated to the same machine which could be a Sun workstation. The Switching Domain object can here be either a physical switch with controlling software like an ATM based experimental broadband switch or simply the necessary simulation software on a computer.

## 4 Concluding remarks

We have in this paper developed a life cycle for IN services and given examples of what a service environment could look like. We have achieved the mapping from the life cycle model to a set of physical entities constituting a service environment by using a simple object oriented approach.

The life cycle we have developed is more detailed than the generic life cycle in ordinary software. It has been possible to do this because we have a specific context (the IN) in which the software is produced. In particular, the response time and availability requirements for services indicate a separation between activation, invocation and execution. We have added the deactivation and deinstallation phases because these are non-trivial in an IN.

One of the important observations concerning the intelligent network is that it is a service-driven, that is, a customer-driven network. The flexibility we want to achieve by introducing the IN introduces some general problems that the software community has been aware of for a long time. These include the following problems concerning the communication challenge between the one who initiates the creation of the service and the one who produces it:

- There is a gap between the subscriber and the service creator. It is therefore essential that the service creator fully understands what the subscriber wants and also that he is able to discuss what a reasonable telecommunications service is.
- The subscriber does not always know exactly what he wants. This implies that the analysis should go very thoroughly through the subscriber's initial requirements so that his set of requirements are actually stabilised.
- The subscriber's requirements to the service might change in the interval between his acceptance of the specification and the activation of the service. This stresses the importance of rapid introduction of services.

This paper is an attempt to refine our ideas about the intelligent network, and it will be suitable as input both to the European Institute for Research and Strategic Studies in Telecommunications (EURESCOM) and to our own work on services and service specification. It will also be fruitful in our efforts to initiate an IN laboratory. There are numerous important aspects related to modelling and implementation of IN not included in our discussion. This is mainly because our primary focus for this paper is the service life cycle. Topics such as service creation, connection control, signalling, protocols, terminal aspects, concurrency, distribution, user interfaces, etc. will demand a much higher level of detailing and are therefore topics for future research.

## Acknowledgements

A draft of this paper was used as input to work on a project in EURESCOM on the evolution of the intelligent network. We would like to thank the subtask leader for the work on the IN service life cycle in this project, Beate Uhlemann from Deutsche Bundespost Telekom, for feedback on this draft.

## References

- 1 Bugge, B et al. *Methods and tools for service creation in an intelligent network, initial document*. Kjeller, Norwegian Telecom Research, 1991 (Research Doc. No. 34/91).
- 2 ETSI. *Intelligent network: framework*. Valbonne, 1990 (ETSI DTR/NA-6001, version 2).
- 3 McDermid, J, Rook, P. Software development process models. In: McDermid, J (ed.). *Software engineer's reference book*, Oxford, Butterworth-Heinemann, 1991.
- 4 Reenskaug, T et al. *OORASS: Seamless support for the creation and maintenance of object oriented systems*. Oslo, TASKON, 1991.
- 5 Wirfs-Brock, R J, Johnson, R E. Surveying current research in object-oriented design. *Communications of the ACM*, 33(9), 104-124, 1990.

# How to handle all the services?

BY HARALD SEIM

654.1:681.3  
621.39.05

## Abstract

*This article gives an overview of the current situation of telecommunication services and describes how advanced technologies are applied to solve the complexity of an ever-increasing number of services in the telecommunication networks and to satisfy the requirements of a more and more market oriented situation for the providers.*

## 1 The current situation

The demand for advanced telecommunication services has increased enormously the last few years. This has led to situations where the network operators have clashed new services into their networks at high speed to satisfy the customer needs. Today, when the telecommunication monopolies are breaking up, the fight for market shares has become hard. New services are marketed as intelligent services and network operators are talking about intelligent networks.

This chapter will handle the service concept as it is today by defining some different types of services and describing some sample services. The role-players in the service market are described to give an impression of who are involved, and the different problems and solutions concerning current services are touched, just to give an impression of the complexity the telecommunication researchers are facing today.

### 1.1 The service concept

The word 'service' has turned out to be a piece of magic in the telecommunication

world the last couple of years. Everybody's attention has moved from switching technology to this word. The service market is expected to grow incredibly in the near future, and the winners of the game are those who can offer the best services at the lowest price. The customers' needs are put in focus, and they shall be satisfied by an irresistible set of fantastic services.

But still, with this enormous focusing on services, this word is somewhat fuzzy and ambiguous. There are so many aspects of services that there is a need for explanation of some of the most used terms.

*Telecommunication services* is a common name for all services offered by, or over, a telecommunication network. It is normally subdivided into bearer services offered by the network, and teleservices offered by the terminals, but it can also be divided into basic services and supplementary services.

*Bearer services* are capabilities for transmission between two points, including physical routing and switching. The services are offered by the network and used by terminals to set up communication paths and transmit information through the network. Examples of bearer services are: circuit-mode speech, circuit-mode audio, circuit switched data and packet switched data.

*Teleservices* include all capabilities for communication between two applications. The services are offered by the communication software in the terminals and used by applications to set up connections and communicate. Examples of teleservices are: telephony, telefax and videotex.

*Basic services* are capabilities necessary to handle basic calls, i.e. call setup and call release. These services are always present when telecommunication services are used and can be viewed as the mandatory part of all telecommunication. There are basic bearer services as well as basic teleservices, and examples of those are mentioned above.

*Supplementary services* are optional capabilities that can be used as a supplement to basic services. These services can only be used together with basic services, and some examples are: calling line identification, call forwarding, and call waiting.

*Value added services* is a term often used for advanced supplementary services, especially services that can be offered and marketed as stand-alone products. Examples of such services are freephone, premium rate and televoting. Many value added services can be offered by special service providers connected to the network.

*Service features* are the basic components of all telecommunication services. The services are built up by different features like e.g. routing, charging, call control, logging, security and management. Different services are normally distinguished by how these features are used. In many ways these features can be seen as services themselves, or at least service components, since it is the sum of these features that are called telecommunication services.

### 1.2 Some service examples

From an intelligent network point of view, the most interesting services are the so-called value added services. They have the largest market potential and can easily be offered as stand-alone products.

This section will first describe the Plain Old Telephony - POT - service, from which most value added services are developed, and then describe some of those services in the light of the POT service. The services described here are only a few examples out of hundreds of new services, some already implemented and some which are coming up in the near future.

*The Plain Old Telephony service (POT)* is based upon a quite simple concept where all users are permanently connected to the network by single telephone lines with only one information channel. This channel is used both for interaction with the network and for conversation with other users.

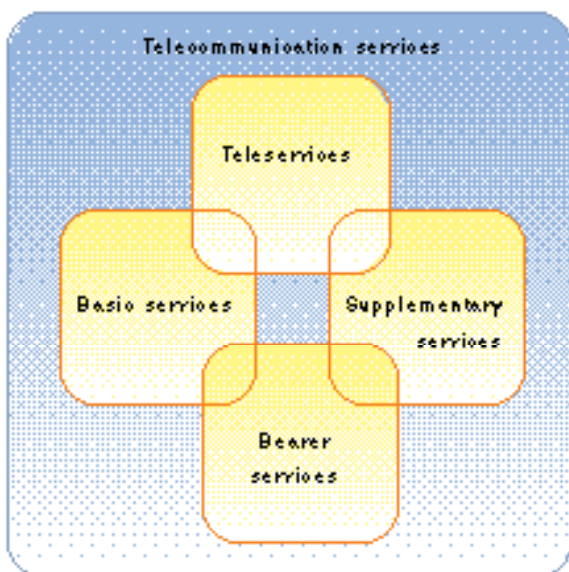


Figure 1 The telecommunication services can be divided into bearer services and teleservices, or basic services and supplementary services.



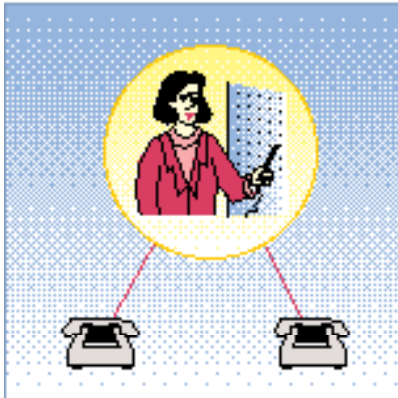


Figure 2 Almost all modern telecommunication services are based upon the plain old telephony service, and the advanced supplementary services are just reinventions of the early days' human operators.

The basic version of this service allows only three phases to be passed in the right order, namely establishment, conversation and release. During the establishment phase, the user gives information to network by dialling a destination number. The number is analysed to find the right destination before the connection is set up. After setup there are no possibilities for further interaction with the network. There is only one path to follow, and that leads to disconnection.

Newer versions of the POT service allow interception during the conversation phase. This is used to give the network new directions in how to serve the user. Other versions are using a separate signalling channel, by which the user can interact with the network during conversation. These techniques are utilised in many new services like e.g. call waiting and call conferencing.

*The call waiting service* is normally provided as a supplement to the POT service. The network informs busy users about incoming calls, i.e. waiting calls, so that they can answer them if desired. Normally, the user must disconnect the existing call to be connected to the waiting call. Sometimes it is possible to ask the network to hold the existing call, so it can be resumed at a later point in time. This last feature requires relatively advanced interaction between the user and the network.

*The freephone service* allows users to make free calls, i.e. make calls without being charged for it. Normally, the called party is charged, but it is also possible to let a third party pay for the calls. Initially, the only difference from the POT service is the charging, but many network operators provide additional features, like e.g. flexible routing, together with this service to make it more attractive.

The freephone service does not require any special interaction between the user and the network. It is more of an internal affair for the network to handle the charging aspects. But, the network still has to be triggered in some way to make the right charging. Today this is done by using a specific area code, also called service code, for the freephone service.

*The premium rate service* is another example of a typical charging service. Here the calling party is given an additional charge for services provided by the called party. These extra services are normally some kind of valuable information. The network provider charges the service users and pays the information providers. This service is triggered the same way as the freephone service, namely by a specific area code.

*The universal personal telecommunication service (UPT)* enables the user to receive incoming calls and initiate outgoing calls from any terminal in any network, based upon a unique number called the UPT number, which is assigned to every UPT subscriber. The UPT number can be registered at any terminal in any network to give personal mobility. Outgoing calls are charged to the UPT number and not to the terminal being used. Based on the dialled UPT number, incoming calls are automatically routed to the terminal where the UPT number is currently registered.

This service involves many advanced features. Both routing, charging and user interaction is much more complex to handle for a UPT service than any of the previously described services. In addition, new aspects like security and management of subscriber information have to be addressed.

### 1.3 The service role-players

The previous section used terms like 'service users', 'service subscribers',

'network providers' and 'service providers'. But who are they, and what is their impact on the future telecommunication services? This section describes the different role-players, their needs, requirements and offerings.

There are mainly two groups of role-players: the customers and the providers. The customers are those who are requesting, paying or using the services, normally called service users and service subscribers. The providers are those who are providing and selling services to the customers, and they are normally grouped into network providers and service providers.

*The service users* are those who are served by the telecommunication network in order to satisfy some need they have. Normally, a user wants to communicate with someone, but in many cases the communication is subordinated to other needs, like e.g. access to information of some kind.

Both the calling party and the called party are service users during a call, regardless of who initiated the call and regardless of who is paying for it.

Today's service users want more and more advanced services, but this also puts more requirements on the service users themselves. Advanced services are difficult to use, and normally this involves complicated interaction procedures and advanced terminal equipment.

*The service subscribers* are those who have more or less permanent needs for being identified by the network, either to be reached by others or to be treated in specific ways during service usage. Normally, the subscribers are paying for permanent connections to the network, from where they can use telecommunication services and be reached by other users.

Today, the subscribers want more and more control of the services they are subscribers of. They want flexible services, which they can customise to fit their specific needs and economies. There are also needs for more flexible network connections. Some subscribers do not want to be bound to a fixed location, but rather be identified by a mobile identity. An example of this are the UPT subscribers.

*The network providers* are those who own, run and maintain the telecommuni-

cation networks, either fixed networks, land mobile networks or satellite networks. The networks consist mainly of transmission systems and switches. In addition, the bearer services are considered as essential parts of the networks.

Until now, most network providers have been protected by monopolies, and have therefore controlled all the services in their networks, including supplementary services and value added services.

*The service providers* are those who offer services to the service users. Until now, the service providers have been identical with the network providers, but in the future, the network providers will only be one of many kinds of service providers. In addition, we will find providers of different value added services as well as providers of basic services based on leased lines.

Service providers not having their own networks, will need some special connections to networks owned by network providers. The European Commission is going to regulate those connections by the Open Network Provision - ONP. These are regulations for the interfaces between network providers and service providers.

#### **1.4 Current problems and solutions**

The current network solutions, where the service software are mixed together with the switching software, and new service features are added on as and when it becomes necessary, are starting to be very complex and difficult to manage. Every new service is treated ad hoc, and the networks and switches are looking more and more like enormous software conglomerates.

Long implementation time is much of a hindrance for introduction of new services today. Lack of modularity implies that all new service software has to be merged with existing software, and for most new services this has to be done in all network nodes, which can be several thousand.

An alternative way of introducing new services is to keep their operation to only a few centralised network nodes. By doing this, implementation time and costs are saved by avoiding updating of all network nodes. A drawback with this solution is that all usage of such services

has to be routed via the specialised service nodes. This implies a very poor utilisation of network resources, since even local calls may have to be routed twice across the network in order to use a service.

When all services are treated differently, and many of them also executed in different nodes, interaction between them may be quite difficult. There will be situations where two services may cooperate, but there will also be situations where simultaneous invocation of two services will imply conflicts.

There will also be a need for pan-European services. If such services are implemented by interaction between national services, one will meet problems with different implementations of the services in different countries. Lack of common specifications, common implementations and common interfaces makes it almost impossible to implement pan-European services by interconnecting national services. Security will also cause problems here, since many network operators are careful concerning access to their data.

With a large number of services, the users are facing a lot of opportunities. But many users find it difficult to use and utilise all these new features. It is difficult to remember how to invoke different services, and the procedures for doing it are often very complicated. Because of this many of the new services are very seldom in use, even though many people would have liked to make use of them. One way of solving this problem, is to have more intelligent terminals. Terminals with specialised function buttons, or menu choices, can make it easier to use the services.

## **2 New approaches**

One of the main goals with the intelligent networks is fast and efficient introduction of new services. Another goal is flexible services, allowing a large degree of customising. Much research effort has been put into reaching these goals and to solve the increasing number of problems concerning telecommunication services. The networks and the services are modelled in new ways and advanced software technologies are applied to manage some of the problems.

This chapter presents some of the research going on in the intelligent net-

works area. The first section describes the separation of network and services. The next two sections describe how object oriented techniques can be applied to modelling of network and services and how rule based techniques can be applied to marketing, creation, administration and execution of services.

### **2.1 Separation of network and services**

Characteristic for the intelligent networks is the separation of the service logic from the physical network. There are different opinions about exactly where this separation should be done, but the main principles are commonly accepted. Regardless of where the separation is made, new services may be introduced much easier since they can be implemented without modification of the underlying network.

The separation of services and switching requires a well defined interface between the two domains. The switches have to send requests and status information to the service logic, while the service logic will have to do operations on the switching network. If the interface is made very general and independent of particular networks and services, the two domains can be changed independent of each other.

#### **2.1.1 Partial separation of network and services**

In the first versions of intelligent networks, the separation is true only for the supplementary services, and even not for all of them. All basic call processing, and some of the simplest supplementary services, will be handled by the switching nodes themselves. Only in more complex situations the control will be handed over to some specialised service nodes. This implies that the basic services and the simplest supplementary services will be handled as they are today. A standard basic call process will be used, and this has to be current in all network nodes. The basic call process will need a trigger table describing the situations for which it has to request help from a service node.

These early versions of the intelligent networks are based on the client/server model where the switching nodes are clients served by the service nodes. It requires that the service nodes are pas-

sive until they are asked for help. This type of separation is quite suitable for services requiring some kind of advanced analysis during call establishment or call release, e.g. for number translation, special charging or conditional routing.

The service logic in the first versions of intelligent networks will only be able to control one end of the call at a time. For this purpose, one basic call process model for the originating end and one for the destinating end of a call have been defined. The two models are used to describe how the basic call process shall behave in different situations at the two ends of a call. The basic call models contain trigger points at which the service logic may be invoked. If new service logic requires trigger points not yet defined, the basic call process has to be modified, and this may have to be done in all nodes all over the network. So, with a partial separation of network and services, it will still be necessary to modify the switches, even when working with services.

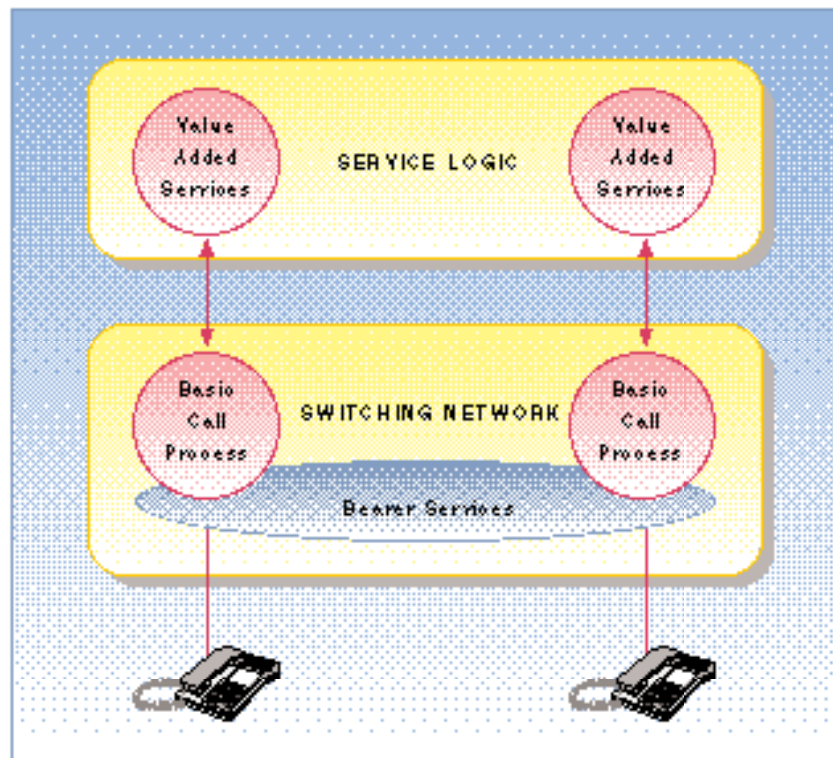


Figure 3 When network and services are partly separated, the basic services and some supplementary services are executed in a basic call process together with the bearer services, and the external service logic is invoked only when the basic call process needs help.

### 2.1.2 Total separation of network and services

In some later versions of the intelligent networks all services, except from the bearer services, are separated from the switching network. The result of such total separation is a pure switching domain and a pure service domain. The switching domain is only concerned with transmission and switching, and offered by this domain are the bearer services. The service domain contains all other services, i.e. both basic services and supplementary services. Then all call handling are done in the service domain, since the bearer services do not include call handling, only capabilities for transmission and switching.

Also in this last model, there will be a natural client/server relation between the service domain and the switching domain, but now the service domain is the client and the switching domain is the server. The switching domain consists more or less of resources controlled by services in the service domain. There are also similar client/server relations between the service users and the services and between different services.

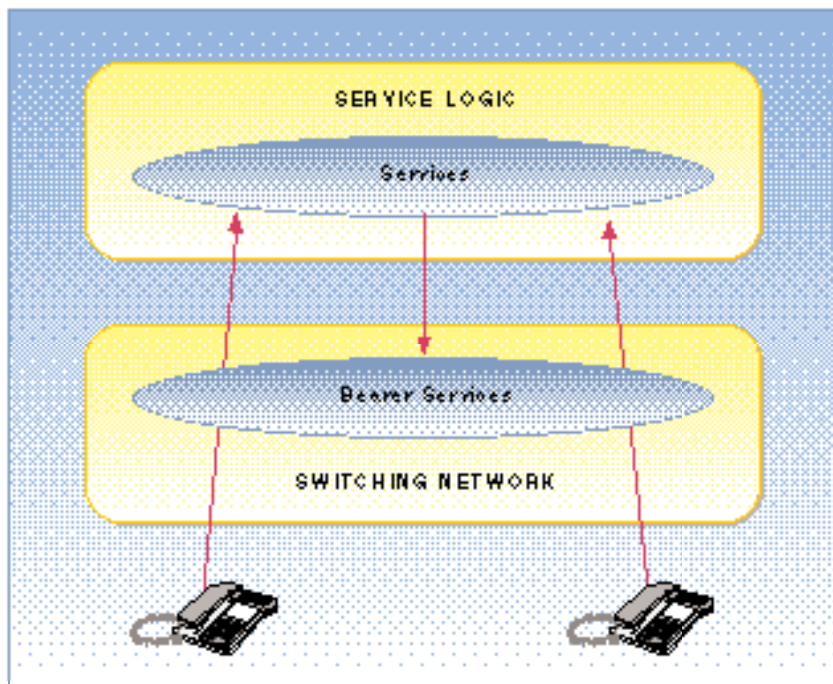


Figure 4 When network and services are totally separated, only the bearer services are executed by the switching network, and the network is acting like a server for the service logic. The users interact directly with the service logic.



### 2.1.3 The IN platform

When the service logic is removed from the switches, there will be a need for a set of common switching operations provided by the network. This set of operations will be used by the service logic to control the resources in the network. The actions taken by the service logic will then be sequences of such operations. Examples of operations for call processing are: join, split, modify, etc.

The switching operations represent a set of network independent instructions to be used by all services. They will make the platform upon which all services have to build, and this platform is called the IN platform.

### 2.1.4 A generic service definition

When all services are taken out of the switches, it will also be easier to treat them in a unified way. To do this, we need a generic definition of what a service is. One definition could be:

*A service may be either a set of capabilities that is offered to an end-user, e.g. conference call, or a set of capabilities that is used in support of these end-user services, e.g. translation of freephone numbers to real telephone numbers.*

With this understanding of what a service is, we can start modelling the service domain.

## 2.2 Object oriented techniques

Object orientation is a way of structuring data and operations in modules called objects. The technique has many advantages to traditional structured programming, and some of the main benefits that can be utilised in the work with intelligent networks are described below.

*Modularisation.* The objects represent modules which can be arranged in ways that are more understandable for human beings. The reliability will increase with increasing understandability. It may also be possible to test and verify an object once and for all, and then rely on that for later use and reuse.

Object descriptions, called classes, can be reused over and over again, and if they do not fit exactly, new specialised subclasses can be defined. These subclasses will inherit data and functionality

from the more generic classes, and by this reuse as much as possible. An inheritance hierarchy will be very easy to maintain, since global changes can be done just by modifying the generic root class, from which all the other classes inherit.

*Encapsulation.* All data are encapsulated by the objects, and can only be accessed by calls to the operations. Encapsulation is done to protect the data by controlled access through operations. Object operations are invoked by message passing. The messages, normally implemented as

function calls, have to be defined with names and parameters. Many systems are also using contracts to restrict the access and to guarantee the results of the operations.

Encapsulation of data will also hide internal differences in the objects. The internal representation of data is unessential as long as the message passing is well defined. Therefore, specification of objects are mainly concerned with the object interfaces, i.e. the operations, and not with the internal structures.

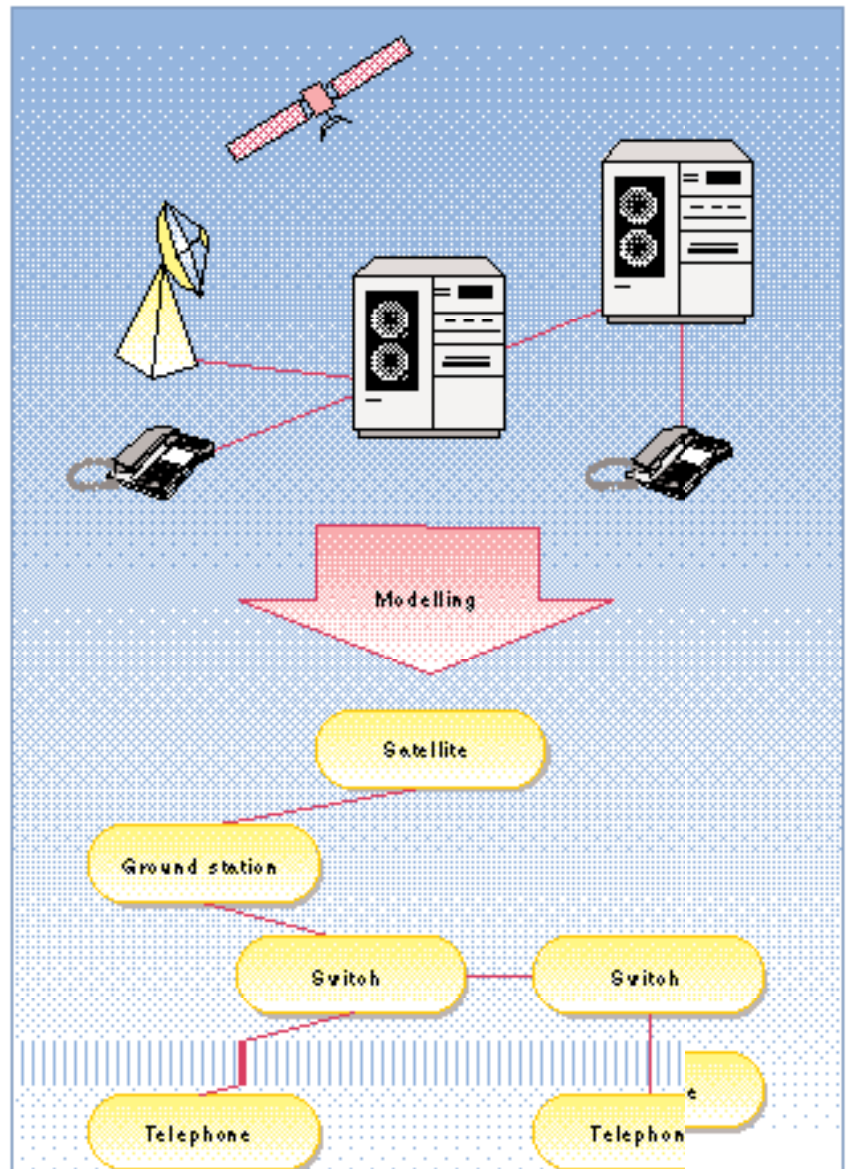


Figure 5 The real network, consisting of various kinds of equipment, is modelled as objects and relations. Each object is an abstract description of a real entity with attributes, operations and relations to other objects.

For telecommunication services, encapsulation can be used for hiding implementation specific solutions and thereby to gain manufacturer independence as well as network independence.

*Dynamic binding.* Dynamic, or late, binding is another feature used by most object oriented systems. It means that function calls are bound to the functions at runtime, rather than at compilation time. This can be utilised by having dynamic objects, i.e. objects that are created, deleted or changed at runtime.

The dynamic binding feature gives a flexibility that can be of great value for provisioning of services. In the service logic, different objects may have to be used depending on e.g. the user, subscriber or even time of day. By dynamic binding, a program's control structure can be changed dynamically at run-time, resulting in more flexible service provisioning.

### 2.2.1 Network modelling

Object orientation has shown up to be one of the most adequate techniques for data modelling. It can be used for modelling of the network resources used by the services as well as the services themselves. When used to model the network resources, the object model can be viewed as a map of the network, showing only the details necessary for service execution.

In the following, some examples of objects representing network resources used for call handling are described.

*The call object* represents a request from a service user. This may be a request for connection or just a request for information. Normally, a call contains information describing what service the user wants, e.g. connection to a given address, quality of service for a connection, etc.

*The call queue object* is a list of calls, i.e. a place where calls can be stored temporarily while they are waiting for something. The call queue may be a special case of a more generic list object, or maybe just of a database object.

*The conference bridge object* represents the resources needed for switching, mixing and presentation of information in a conference with several users. The object should be able to handle both plain tele-

phony conferences and multimedia conferences, e.g. voice, video and text.

*The connection object* is an active communication path between two service users. A connection may be a set of legs interconnected by connection points.

*The connection point object* represents an interconnection of two or more legs.

*The counter object* represents a generic counter. It can be used for counting of calls, etc.

*The database object* represents a storage place for information. It should have an interface with functionality for authentication, batch reporting, dialog management, electronic mail, job control, graphics, ordering, reorganisation, etc.

*The information object* represents information provided by the network. Information objects may contain announcements, directory entries, mail, user dialog boxes, user profiles, etc. Information may be stored as data, picture, text, video, voice, etc. Information objects may be stored in databases and transmitted over connections.

*The leg object* represents a communication path between two addressable entities. Two or more legs may be interconnected by a connection point.

*The subscriber object* represents a service subscriber, including all subscriber specific data, or at least references to them. Examples of subscriber specific data are: service profiles, user profiles, etc.

*The timer object* represents a generic timer which can be used for measuring of call duration, automatic disconnection, etc.

### 2.2.2 Service modelling

By an object oriented analysis we can break down the services to their basic components. The set of service components will then cover all capabilities necessary to offer the current services. If we define the components so they can be used as building blocks for services, they can be reused in a large number of services just by putting them together in varying combinations.

The first step in a top-down analysis of the services should be to define a set of

generic objects covering all the main service features. Then, more specific features can be defined by subclassing the more generic object classes. A first top-down analysis of current services could end up with the following objects:

*The charger object* is responsible for all charging of service usage. The following types of charging may be done:

- Call state dependent charging
- Geographic dependent charging
- Normal charging
- Premium rate charging
- Reverse charging
- Service dependent charging
- Split charging
- Third party charging
- Time dependent charging.

*The call controller object* is responsible for handling of all calls from the service users. It may be subclassed into e.g. basic call controller, conference call controller, etc. The following functionality must be covered:

- Call queueing (automatic call back, number in queue, priority, queueing time estimation, etc.)
- Call set-up (quality of service, e.g. charging, priority, transmission, etc.)
- Call release (automatic (call duration limit) or manual)
- Call transfer
- Call waiting
- Completion of call to busy subscriber
- Completion of not answered calls
- Conference calling (add-on, leave, meet-me, three-party, etc.)
- Delivery to alternative destinations
- Explicit call transfer
- Hold/retrieve
- Suspend/resume.

*The logger object* is responsible for logging of all relevant information. The following functionality should be covered:

- Call counting
- Call logging

- Subscriber statistics
- Traffic measurements.

The *manager object* is responsible for all management activities. This includes the following:

- Billing (specified bill, etc.)
- Business management
- Network element management
- Network management (accounting, configuration, fault, performance and security)
- Service management
- Service profile management
- Service profile verification
- Subscriber administration.

The *router object* is responsible for routing of all calls, messages, requests, packets, etc. between different network entities. The following functionality should be covered:

- Abbreviated dialling
- Alternative destinations
- Call barring
- Call blocking
- Call deflection
- Call distribution
- Call diversion
- Call forwarding
- Call gapping
- Call limiter
- Compatibility handling (service/terminal dependent)
- Direct dialling in
- Distribution (list)
- Electronic mail (text)
- Follow-me diversion
- Hotline, direct or delayed
- Line hunting
- Mobility
- Number translation
- Prioritising
- Queueing
- Restricted area covering
- Sub-addressing
- Traffic filtering
- Voice mail.

The *securer object* is responsible for all the security in the network. The object should guard both the network's data and the transmitted information from unauthorised access. It should be possible to execute services with different security level, according to the needs of the network operator, the service provider and the service user. The security object should cover the following functionality:

- Access control
- Authentication
- Authorisation
- Calling card validation
- Closed user group
- Confidentiality
- Encryption
- Screening
- Verification.

The *user object* represents a user of a telecommunication service. A user may be a person, a computer program or a combination of these two, i.e. what normally is called an application. A user object may request a service by creating a call and send it to a user agent object. The user object is responsible for the following:

- Call answering (accepting)
- Call initiating

- Call rejecting
- Call state changing, e.g. disconnecting, holding/retrieving, rearranging, releasing, suspending/resuming, transferring, etc.

The *user agent object* is responsible for all interaction with service users and is therefore an essential part of the user/network interface. The following functionality should be covered by this object:

- Advice of charge
- Announcement
- Calling line identification presentation/restriction
- Connected line identification presentation/restriction
- Dialog management (including alphanumeric and graphical dialogs)
- Directory
- Distinctive ringing
- Information collection
- Prompting.

### 2.3 Rule based techniques

Rule based techniques have until recently only been a part of the mysterious domain of artificial intelligence. But, during the last years, rule based techniques have been approved by many excellent implementations, and are now

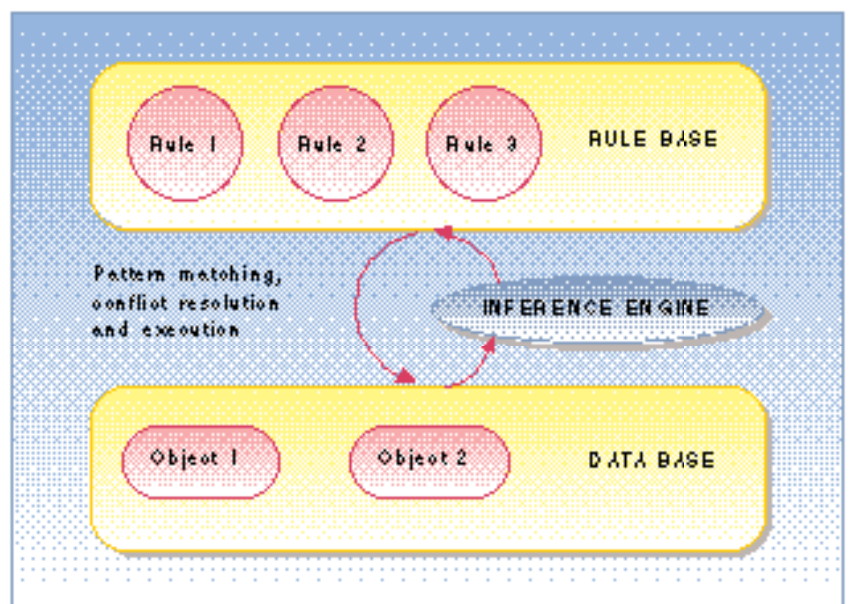


Figure 6 The main components of a rule based system are the database, the rule base and the inference engine.



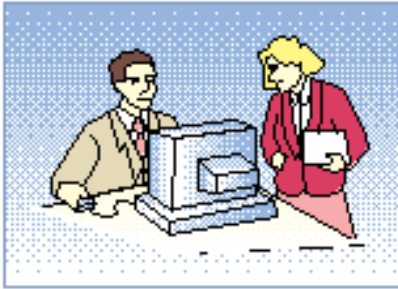


Figure 7 Expert systems will be of great help for both service sales people and service designers in their contact with customers. The systems can be used to give information, analyse needs and suggest solutions.

widely used by the telecommunication industry. The most common use of rule based techniques is in expert systems. There heuristic rules are used for implementation of expert knowledge. The main components and features of rule based systems are described below, and in the following sections some applications of rule based systems are introduced.

*Data* represent an essential part of every rule based system. Data structures are used to make models of the real world or to store temporary information. The data in a rule based system are often called 'facts'.

Data can be represented in many ways. Frames have been used in many implementations. Frames may be viewed as a kind of objects. They offer modularity and inheritance, but they have no encapsulation and no operations. Frames are well suited for symbolic data, which is the main data type in rule based systems.

*Rules* are modules used for representation of logical knowledge in a system, i.e. knowledge about how to use the data, when to take different actions and where to find necessary information. All rules have a set of conditions and a set of actions. The actions describe what to do and the conditions describe when to do it.

Conditions are logical expressions that can be evaluated to either 'true' or 'false'. Symbolic data are used in the expressions. If all conditions of a rule evaluate to true, then the actions in that rule can be executed. The actions can be operations on the data, and if the data are changed, then the set of rules that can be

executed will change continuously. This is called data driven control.

An *inference engine* is used for evaluation of conditions and triggering of rules. This generic inference mechanism will provide domain specific reasoning when applied to a set of domain specific data and rules.

When the inference engine is running, it follows a cycle with pattern matching, conflict resolution and rule execution. The inference engine keeps the data on a blackboard, so they can be accessed by all rules. Common data on the blackboard represent the only coupling between different rules. This weak coupling between system modules gives rule based system a high degree of flexibility.

### 2.3.1 Service marketing

Even today, the service marketing and sales people need access to a lot of technical information in their contact with the customers. When the intelligent networks are being implemented, the increase in number of services and customised solutions will be enormous. The marketing and sales people will need support to find the best solutions, estimate prices and costs, give proposals and answer questions. System specifications and manuals will be of great help, but even with all information available, one will need much experience to find the relevant information and to know how to use it.

Much of the expertise needed for marketing and sale could be stored in an expert system. Such a system could help the marketing and sales personnel to find the relevant information, and it could use information gathered from the customers for analysing needs and giving recommendations. Some examples of what expert systems could be used to, are:

- Analysis of service needs
- Support in describing new services
- Information about existing services
- Information about possible services
- Information about costs
- Information about equipment requirements.

### 2.3.2 Service creation

Service creation is a very knowledge intensive process where rule based systems could be of great value. A service

designer will need a lot of information about network configurations, possibilities and limitations, existing services and the subscribers. A rule based expert system can give easy access to all this information.

The service creation process will also require a lot of intelligent reasoning, where expert systems could be used. This can be for selection of service modules and for right combination of them in order to get the desired service.

### 2.3.3 Service administration

Service administration will become a heavy job in the intelligent networks. Service logic will be distributed in the network according to needs, and the distribution will be done in real time, i.e. during service execution. This implies an automatic distribution based on a set of rules for how to distribute. The distribution has to take into account things like performance, response times, resource utilisation, etc.

Another service administration problem is the integration of a large number of databases. Service logic and service data has to be stored in databases all over the network, and even over many networks. All these databases must cooperate in the provision of services. Some services may need information from several databases, and both logic and data has to be moved around between different databases. There are expert systems on the market which are designed especially for integration of databases, and from which ideas can be taken to solve the database integration problem also in the intelligent networks.

### 2.3.4 Service execution

Today, most services are implemented as sequential functional programs. This gives very little flexibility and few of the benefits described for object oriented and rule based techniques in this article. It is therefore expected that features from both these techniques will be used in future developments. Object oriented techniques have already been introduced to some current systems, while rule based techniques still remain in the prototype stage.

Some of the benefits of using rule based systems for service execution are:

- Rules are well suited for taking complex decisions.
- Rules are well suited for handling symbolic data, like e.g. object oriented network models.
- Rule based systems are data driven and the program control changes with changing data.
- New services can be developed evolutionary by stepwise prototyping and simulation.
- Rules based on conditions and actions are well suited for describing services. Human reasoning can easily be expressed and many service interaction problems can be solved.
- Rule based systems are easier to maintain because of the modularity and the loose bindings between different rules.

The best way of implementing complex systems like service logic and service execution environments is probably to combine several techniques using the best features from all of them. Currently, the object oriented and rule based techniques are the most promising, together with conventional programming techniques.

# Description and specification of services in Intelligent Networks

BY ERIK A COLBAN AND KLAUS GAARDER

## 1 Introduction

A public network operator offering Intelligent Network (IN) services to its customers must be able to describe the services not only to their users, but also to the subscribers of the services and to the service and equipment providers. A subscriber can customise a service to her specific needs by setting values to different parameters or selecting features. This implies that the subscriber needs a more detailed and deeper understanding of the service and its components than the user. The differing needs of the user and subscriber must be reflected in the descriptions aimed at each of them. A description aimed at the service and equipment provider will have yet another focus. In this case, the role of the public network operator is to specify a product. Implementational details are not always critical. However, the specification must be precise enough to allow for a quality evaluation of the product.

In order to operate services on an international scale across different networks and have services adapt smoothly to technological changes, services must be described independently from any specific network. This should be the case whether the descriptions are aimed at the users, the subscribers or the providers.

The requirements above are specific to IN service description. In addition there are the usual requirements that every description or specification should meet, such as precision, coherence and verifiability. In the following we shall discuss how the latter can be met in the context of IN.

Services cannot be described entirely in a natural language. In a final section, criteria for appropriate choices of description languages are discussed.

## 2 Goals

Any description or specification should be precise. To ensure a common understanding of "precision" we quote the following interpretation from a standard dictionary:

*"precision exactness and clarity; quality of being precise (stated clearly and accurately)."*

Note that *precision* embodies not only exactness but also *clarity*. Clarity and exactness may be perceived as con-

flicting requirements, since a higher degree of exactness often means a more involved description, which may reduce clarity. This is an important issue for us and we must take care to cater for this as we develop our methods. We may talk about *differing levels of precision* being right for different uses and thus justify varying descriptions to obtain maximum clarity on each level.

A description is coherent if its parts fit naturally and logically together. If a term is not used with the same meaning in all parts of a description, it may become difficult to see how the parts relate to each other.

In order to assure coherence in the description of IN services, a set of concepts that are particular to the subject need to be defined. An attempt to define these concepts is found in the CCITT Q.1200 Recommendation Series, see (9), which draws a picture of the IN concept and some of the services that it supports. In our opinion, these definitions are too rough and may cause divergent understandings of the concepts. Since several persons usually collaborate in describing the services, coherence is at stake.

A service can be given different descriptions, each focusing on different aspects while abstracting others away. This does not necessarily mean that there is incoherence as long as the different descriptions are related to each other in such a way that they complete each other and give a consistent view of the service.

A specification constitutes a reference against which a product is evaluated.

*Definition 1 (Quality)* A product has the right quality if and only if it is according to specification.

When strictly defined quality is binary term taking values "right" or "wrong", not "good" or "bad" or something in between. This may seem a strange way of judging quality, but it is the only verifiable or quantifiable way. Quality concepts which are not amenable to verification or testing are useless in technical environments, a specification constitutes a protection for producer and consumer alike. Ideally the consumer knows what to expect from the product and the producer knows what she must supply. It is the consumer who supplies the producer

with a specification of the desired product. If the producer then delivers a product which is according to the specification, she has done her job. The consumer cannot *expect* more and the producer need not *supply* more, than stated in the specification. *Verification* consists of *checking that a product is satisfying a given specification*. If the product is e.g. a computer program, it can never be "correct" in and of itself. Correctness in this context is only defined relative to a specification. Section 5 discusses this matter in further detail.

## 3 Precision

Specifications are written to be used as input to a production process where the result depends explicitly on the specifications entering the process. In our case this is the IN service creation process. The term *service creation* is used to designate the process from specification up to and including deployment of the finished service.

A small difference in perception of meaning in a description may imply very large differences in the characteristics of the end product, which is undesirable. If the process towards the finished product is long and involves many links in a chain, small imprecisions in the initial phases are likely to grow larger as they pass through the chain.

In an object oriented environment, object interfaces must be according to specification in order for objects to be able to cooperate. In the specification of classes this results in a specification that is "open ended". That is, it does not specify every detail, only what is necessary at the interface. A specification may thus allow *various implementations* which are all correct, since the specification leaves some decisions open. Precision is nevertheless required as before in the interface specifications and the further specification of the interior of classes.

Our goal is to make IN service creation

- fast
- flexible
- reliable.

We believe that service description and specification are two main issues in achieving these goals, and that successful

621.39.05  
681.324  
681.3.01



solutions to these issues will be of crucial importance.

In the intelligent network we aim for high precision and formality in specifications which will be created by people with various training on many levels of sophistication. This is the novel challenge posed to us by the intelligent network concept. Below we shall further concentrate on three key characteristics of specifications:

- readability
- clarity of exposition
- unambiguity.

We shall discuss each of these in turn.

### 3.1 Readability

The readability of a description is obviously crucial for how successful it is in bringing across the intended meaning. Whether it is *considered* readable depends on the profession of the reader. Law text is readable to lawyers and mathematical text to mathematicians but not necessarily vice-versa. This is of no problem since lawyers mostly stick to law and mathematicians to mathematics as professionals. The challenge of IN service creation is that specifications must be readable yet precise, to a quite wide audience and across many levels of abstraction and sophistication.

Attaining readability can be done in many ways, and possibly by different means for different readers. The structure of a specification is probably one of the most important parameters deciding readability. A poorly structured document is more difficult to read than one with a good structure. It is not obvious that the same structure is good for all kinds of documents, so we will have to consider the various kinds separately. The structures will have to serve different purposes at various levels of detail and formality.

The structure of a *formal* specification is of special importance. By its nature a formal specification has to have a strict structure from which we cannot deviate, and this structure is a part of the formality of the method. When we choose a formal method we also choose its way of structuring specifications. The technique must have available structures which achieve the wanted readability. For examples of different structures we refer the reader to e.g. (3, 5, 8, 1).

### 3.2 Clarity

Even if clarity may be seen as an aspect of readability it has its own particular needs. We shall look into two which we see as especially important, the use of *definitions* and *symbols*.

Definitions are crucial in any description which aims to be precise and consistent. Many descriptions become cluttered and unclear because the central concepts lack a good definition. Worse still, a concept may be ill-defined such that its extension is vacuous or trivial. Definitions are the means to help us fix and delineate our concepts. For any discussion of concepts or description of concepts, the definitions form the basis from which all further reasoning proceeds.

Consider as an example the *service* concept of the IN. We talk freely about *service* creation, *service* features, *service* interactions, etc., the *service* concept permeates the entire discussion of IN. It becomes necessary to consider a definition of the service concept. Note that this does not mean a *narrowing* of the service concept's extension, since our intention in the intelligent network is a *wider* extension, but always retaining a precise understanding of exactly what it includes.

Following the object oriented line of thought we could start by defining so-called "service-constituents" from which we aim to build other services, and then define what it means to put these together. One possible definition of a *service* is then "whatever is put together by the (correct) use of service-constituents". We have then moved the problem of defining complete services to the problem of defining its constituents and the way these are put together. Hopefully, this will make the problem more tractable.

The use of symbols should be restricted to obtain clarity without throwing readability out of the window. A document packed with symbols tend to appear impenetrable or unduly compact, especially if each symbol carries a lot of meaning, which they tend to do. Used with care symbols are of great value when writing specifications. They let us use a more compact language and thus reduce the chances of misunderstandings and errors following from a verbose presentation. When good definitions have been made of concepts, then symbols can

be introduced to denote instances of objects in the extension of these concepts. Used with care symbols are of great value to us, and in formal specifications they are key ingredients. The success or failure of a formal specification language may rest with its use of symbols. This is particularly true when the language is to be used by people without heavy mathematical training.

### 3.3 Unambiguity

Ambiguities in specifications invariably lead to faults and products which are not exactly as the specifier intended. Note that a product may well be said to be conforming to an ambiguous specification, if it is conforming to one of the possible interpretations of the specification. People seem to have a strong capacity to arrive at what they *think* is a reasonable interpretation when faced with an ambiguous specification.

Lives may or may not be lost if the specification of an intelligent network service is misinterpreted, but time and money will surely be at stake. Ambiguity is not an easy thing to control and its source is often in the initial phases of specification where ideas are transformed into specifications. We may not even be aware of the ambiguities in our own ideas. This initial fuzziness can never be completely removed, and its presence is a part of the creative process. However, once a decision is made to go forward and turn the idea into a product the unwanted ambiguities must be removed.

According to a standard Oxford dictionary *ambiguity* means

"presence of more than one meaning".

Thus, to remove ambiguity we need to ensure that concepts have only one meaning, if we want them to have only one meaning. Natural language draws some of its power from the fact that it has an abundance of words which have different meaning in different contexts and usages. This is often not perceived as ambiguity since it is a natural part of our language, but it makes unrestricted natural language unsuitable for many description and specification tasks. Ironically this feature of natural language is perceived as enriching the expressive power of natural language while it can be

devastating in specifications. How can we cope with ambiguity without losing expressive power?

One way of removing most of the ambiguity is by increasing formality. In a formal language we have the power to decide exactly what each expression is to mean, and may thus ensure that each expression has only one well-defined meaning. One of the benefits is that even complex expressions have a meaning which is defined in terms of the meanings of its components. A simple example is an expression like  $A \wedge B$  which has a meaning decided by the meaning of  $A$  and  $B$  and  $\wedge$ . Precision is really all about semantics, the syntax is merely helpful or not in the textual layout. By attempting to formally define the semantics of a language we may uncover sources of ambiguity in the language itself.

## 4 Coherence

In an incoherent description it is problematic to see how different parts relate to each other and the description is difficult to understand. Attaining coherence in the description of IN services is particularly challenging, because services necessitate several descriptions aimed at different people and serving different purposes that need to be related to each other in a logical way.

Coherence requires precision. Vague terms may be understood differently and used with different meanings in different parts of a description, making it difficult to relate the parts to each other.

Terms must also be used in a manner that is consistent with their definition throughout the description. If a term is already used in an inconsistent manner, any attempt to make it precise will result in a disagreement between definition and use.

Consider the term *call* used by CCITT in describing a considerable set of services. It is defined (in Rec. E.600) as:

“A generic term related to the establishment, utilization and release of a connection. Normally, a qualifier is required to make clear the aspect being considered, e.g. *call attempt*.”

This careful formulation seems to intentionally keep the definition vague. The reason, presumably, is that it is dif-

ficult to give a definition that covers all uses of the term *call*.

### 4.1 Missing concepts

Missing concepts often result in gaps in a description, which are another source of incoherence.

The semantics of a service is provided by its functional description, stating *what* the service does rather than *how* it does it. In the IN model described in (9), Q.1202, the “Global Functional Plane” is intended to provide (the concepts necessary for providing) functional descriptions of services. How a service is executed is taken care of on another “plane”, the so-called “Distributed Functional Plane”. Here the execution of a service is described as a set of processes running concurrently on different “functional entities”. However, the way the Global Functional Plane is described today, service descriptions are not amenable to any precise description without resorting to the Distributed Functional Plane. This means that in order to get a complete picture of a service’s functionality, one has to analyse it as a set of different processes running on different functional entities. The notions necessary to give complete functional descriptions of services are missing in the Global Functional Plane (contrary to what seems to be the intention). What remains after distribution aspects have been stripped from the service descriptions is so little that it becomes difficult to find coherence.

The reason that it is not possible to give a precise functional description of services that are entirely contained in the Global Functional Plane is that a description of the network is missing. On the Global Functional Plane the “IN structured network is viewed as a single entity”, see (9), Q.1203. However, no description of such a network is provided. Since it is not possible to give a description of a service without involving the network, and since (9) only gives a description of the network on the Distributed Functional Plane, the functional descriptions have to resort to this plane.

A “functional network”, generic enough to encompass all networks that the IN concept is intended to be applicable to, such as mobile or fixed, line switched or packet switched, connectionless or connection-oriented, etc., has yet to be

defined. Such a network should capture the universal character of a network. A common characteristic of networks is that they allow for information transport. Connections and their end points are also common to all networks. Technology dependent aspects such as transmission speed should only be reflected in the functional network insofar as they are needed in describing the semantics of a service. The transmission speed could, e.g., be reflected by a parameter with no specific value assigned to it.

### 4.2 Coherence through mathematical rigor

In order to provide the semantics of IN services in a coherent manner, a set of concepts that we shall hereafter refer collectively to as the “functional network” must be defined. Genericity constraints imply that the concepts are abstract in the sense that they do not refer to any specific network. Abstraction does not imply vagueness, however. We view the relation between the functional network and the possible realisations of it, as similar to the relation between a mathematical concept, described by some theory, and the specific models satisfying the theory. Figure 1 explains this analogy. The relation between the functional network and the possible realisations of it should be compared to the relation between the notion of a *group* and the different models that satisfy the group axioms such as  $(\mathbb{Z}, +)$ ,  $(\mathbb{Q}, +)$ , ... By adopting mathematical rigor in the description of the functional network, we believe we can satisfactorily handle abstract concepts.

### 4.3 Coherence through object orientation

Throughout this article we advocate an object oriented approach to service descriptions. From a coherence point of view, there are two major reasons for adopting object orientation. First, there is the *reusability* aspect. If two descriptions use the same classes at least some coherence is achieved. Object orientation supports reusability particularly well due to its *encapsulation* and *inheritance* properties (for a presentation of the virtues of object oriented programming, see e.g. (7)). Second, object orientation allows different “viewpoints” to be related to each other in a well-defined manner. Consider a distributed database. From a

functional point of view a database can be seen as one single object to which queries are presented. A distributed view will show several interrelated objects which are capable of handling concurrent accesses, maintaining consistency, etc.

These two views, functional and distributed, are reconciled by encapsulating distribution aspects within the object representing the functional view. Using *deferred classes*, i.e. partially defined classes, one could even describe the functional view postponing distribution matters to the description of some subclass of it (i.e. a class that inherits the deferred class). The reader is invited to compare this example with the one presented in Figure 1.

## 5 Verifiable services

Time consuming work on specifications is of little use unless the specifications obtained provide some real benefits. Once we arrive at a product it remains to see whether it is according to the specification, if it has the *right* quality. The actual process of checking this is called *verification*.

In different contexts it is of varying importance to be absolutely assured that a product is according to its specification. This will always depend on how sensitive we are to an eventual failure. If the product is your hat you have not even thought of a "hat failure", but if it is a piece of software controlling the engine in an aeroplane we have vivid imaginations of what results a failure could have! These are two extremes and we have a broad spectrum in between. The need for verification becomes stronger as we move towards the "engine failure" end of the spectrum. The intelligent network services will be distributed over this spectrum, with a "medium" to "high" placement as typical. The reason is that the failure of telecommunications services is undesirable in most cases and can be catastrophic in some. We should therefore strain to keep the chances of this happening to a minimum. This is partly achieved through good specification and verification work. In fact it is often stated that the most efficient use of resources for quality control is in the specification and verification phase.

An interesting aspect of verification is appearing if we consider two distinct aspects of the specification, namely the

- functional properties, and
- non-functional properties.

By non-functional properties are meant properties which do not relate to the presence of functionality but to properties of the functionality itself when present. Examples of non-functional properties are

- security
- reliability
- persistence of data, etc.

In the case of security we can see that a functionality may be present and be secure or insecure relative to a chosen security policy. If the specification says that "operation  $f(x,y,z)$  should be there" with pre- and post-conditions in order and nothing more, a verification could proceed to give an "OK" even if the operation was totally unacceptable security-wise.

### 5.1 Methods of verification

Different methods of verification may be appropriate for different purposes and properties. We can name four broad classes of such methods, these are

- simulation
- lab testing
- field testing
- formal proof.

Simulation may be appropriate when the final system or product is so large/complex or expensive to manufacture that a verification of the final system is impractical. Simulation will then give a picture of the most important behavioural features of the system, if the simulations are themselves correctly executed. This can seem to move the verification problem onto the simulation itself, and simulation alone can never give a completely satisfying verification and has to be used in conjunction with other techniques.

Laboratory tests are also used in addition to other verification techniques. Unlike simulation (which may also take place in a laboratory) laboratory testing is typically done on a version of the actual product, often a prototype. Apart from this,

simulation and laboratory testing may look very similar and serve many of the same purposes.

Field testing is a way of getting to know how the final product or a prototype behaves in the environment where it is meant to function. This can typically be one of the last stages in a verification process incorporating also simulation and laboratory testing.

Formal proof is perhaps the most controversial of these methods. It consists of actually *proving by formal mathematical techniques* that a proposed product is according to specification. This has as a prerequisite that the specification is also a *formal specification*. Formal verification must be said to be in its infancy with respect to industrial applications but there is a tendency towards more use of formal techniques in the computer related industry and the telecommunications industry in particular. The reasons are among others a need for reliable and safe systems, like the intelligent network. IN services can be subjected to all of these types of verification.

### 5.2 Formal verification techniques

Formal verification derives its name from the fact that the verification consists of deriving a formal proof of statements of formal logic (see the box on formal proof, Figure 2). It is inseparably connected to formal specification and cannot be done with respect to informal specifications.

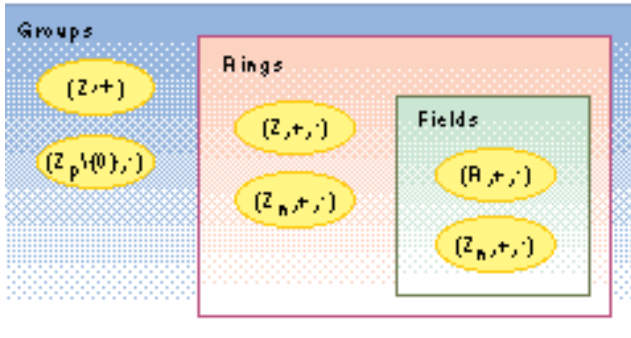
Hitherto formal verification has been closely connected only to computer programs. The reason is partly that computer programs are quite easily considered as mathematical or formal objects amenable to formal reasoning. This was first realised by Turing in 1936, and has since been of vast importance to the development of computer science and programming. In 1969 C.A.R. Hoare in (6) formulated a mathematical theory of programming and program correctness which has since been used extensively as a basis for formal verification techniques.

Simulation, testing and field trial can at best give us an increased probability that our program is conforming to the specification. We can never by use of these methods get an unequivocal "Yes" or "No". The ambition of formal verification in combination with formal speci-



## The Algebraic Analogy

Algebraic structures can be ordered according to their “genericity level”.



Every *field* is a *ring* and every *ring* is a *group*. A typical group is the set of integers with addition, denoted  $(\mathbb{Z}, +)$ , whereas the set of real numbers with addition and multiplication, denoted  $(\mathbb{R}, +, \cdot)$ , is a field.

Groups, rings and fields are defined by sets of axioms. A structure  $(G, o)$  is a group if and only if

1.  $o$  is *associative*, i.e.  $(aob)oc = ao(boc)$ , for all  $a, b, c \in G$  ( $\in$  reads ‘element in’),
2. there exists an *identity element*  $e$  in  $G$ , i.e.,  $aoe = eoa = a$ , for all  $a \in G$ ,
3. every element has an *inverse*, i.e. for every  $a \in G$  there exists an element  $a' \in G$  such that  $aoa' = a'oa = e$ .

For the definitions of rings and fields the reader is referred to any textbook in algebra, e.g. (2).

The set of groups can be described in an object oriented language (here Eiffel, see (7)) by a class *GROUP*.

```

deferred class GROUP
  -- class representation of group  $(G, o)$  with identity element  $e$ 
export
  add, identity, inverse
feature
  add(x,y:ANY):ANY is
    requires  $x, y \in G$ 
    deferred
    ensures  $add(x,y) = x \circ y$ 
    end -- add;
  identity : ANY is
    deferred
    ensures  $identity = e$ 
    end -- identity;
  inverse(x : ANY) : ANY is
    requires  $x \in G$  is
    deferred
    ensures for all  $x \in G$  :  $x \circ inverse(x) = inverse(x) \circ x = e$ 
    end -- inverse;
end -- class GROUP
  
```

The **features** are **deferred** because it is not possible to further specify these features without losing some of the genericity of

the group concept. The features are made precise, however, by the assertions expressed as comments and **requires** and **ensures** clauses. On the basis of *GROUP* a programmer can, e.g., write a program that checks whether a given element  $a$  of a given group is of order 2 (i.e.  $a \neq e, a \circ a = e$ ), and even *prove* that the program is correct!

An instance of *GROUP* can only be obtained by instantiating a non-deferred class that inherits it. Such a class is *ZpGROUP* which describes the groups  $(\mathbb{Z}_p \setminus \{0\}, \cdot)$ , for  $p$  a prime number. (Note:  $(\mathbb{Z}_p \setminus \{0\}) = \{1, \dots, p-1\}$ )

```

class ZpGROUP
  -- class representation of group  $(\mathbb{Z}_p \setminus \{0\}, \cdot)$ , where  $p$  is a prime.
export
  add, identity, inverse
inherit GROUP
feature
  p : INTEGER;
  Create(m : INTEGER) is
    requires  $m$  is a prime.
    do  $p := m$ ;
    end -- Create;
  add(x,y : INTEGER) : INTEGER is
    requires  $1 \leq x, y < p$ 
    do  $Result := (x * y) \bmod p$ ;
    end -- add;
  identity : INTEGER is 1;
  inverse(x : INTEGER) : INTEGER is
    requires  $1 \leq x < p$ 
    do  $Result := x ** (p-2) \bmod p$ ;
    end -- inverse;
end -- class GROUP
  
```

An instance of  $(\mathbb{Z}_7 \setminus \{0\}, \cdot)$  is obtained by declaring a variable  $z_7$  of type *ZpGROUP* and then creating it:

```
z7: ZpGROUP; z7.Create(7)
```

Since *ZpGROUP* inherits *GROUP*,  $z_7$  is also an instance of *GROUP* and thereby it obeys the assertions of *GROUP*.

This example shows:

1. How abstract mathematical concepts can be mapped onto classes defined in an object oriented language;
2. How mathematical descriptions complement the descriptions of the classes. The assertions of *GROUP* refer to the mathematical definition of a group, and would have no meaning if the group axioms did not exist;
3. How the object oriented descriptions rely on theorems based on the mathematical description. For instance, the feature *identity* is defined as a function, relying on the fact that there is only one identity element in a group. This is not stated by the group axioms directly, but is a logical consequence that can be proven. Also, the correctness of *ZpGROUP* relies on the fact that  $(\mathbb{Z}_p \setminus \{0\}, \cdot)$  actually is a group when  $p$  is prime. Remove the requirement that  $p$  (and  $m$ ) must be prime and the description above is no longer correct!

Figure 1

fication is to provide a means of obtaining such answers. We can outline a typical verification technique as follows. We consider a piece of a program,  $P$ , which is to be verified against a specification. This specification will often be of a form saying essentially what is assumed to be the case before  $S$  is executed, the “precondition”  $P$ , and then what is guaranteed to hold after the execution of  $S$ , the “postcondition”  $Q$ . If  $S$  is written in a suitable formal notation the idea is that a formal proof of  $Q$  from  $P$  and  $S$  constitutes a verification of the fact that as long as the precondition holds,  $S$  will guarantee that the postcondition is fulfilled. If the logical system used is consistent this will be a proof of a true statement and hence a formal mathematical proof of the fact that a program is doing what the specification says it should do.

A problem with formal verification is that as programs and specifications become larger the complexity of doing a formal proof explodes<sup>1)</sup>. For this reason formal verification tends to be restricted to programs with extreme requirements imposed on them. This can be the case for e.g. security and safety critical software in the intelligent network.

### 5.3 Formal verification and object orientation

It is interesting to investigate the possible benefits of object orientation on formal specification and verification. The most obvious idea can be sketched as follows: Since object orientation promotes encapsulation of operations related to the same concept into objects, we could benefit from specifying and verifying objects and object aggregates separately. The idea is that building larger structures from proven substructures ought to result in a proven structure as long as the construction process is designed such as to maintain the correctness at all times. This is an interesting research challenge to be pursued further.

To conclude we can say that verification of intelligent network services is important for many reasons, but the most

<sup>1)</sup> The increase in complexity is more than exponential, and generally results in undecidability.

### Formal Proofs

Formal proofs as the name implies, are proofs based on the syntactic *form* of propositions (or statements), abstracting from their semantic content. A formal proof is correct if it can be decomposed appropriately into parts, each part matching one among a fixed set of patterns. Because syntactic form is easily representable in a computer, whereas semantic content is not, formal proofs are particularly well suited to be executed and verified by computers.

The following three lines is an example of a *deduction*, the two first lines are the *premises*, the last is the *conclusion*.

Every man is mortal.  
Socrates is a man.  
 $\therefore$  Socrates is mortal.

The deduction is provable in a formal proof system containing the following rule (or pattern).

Every X is Y.  
Z is an X.  
 $\therefore$  Z is Y.

The proof consists of matching the deduction with the rule, matching X with ‘man’, Y with ‘mortal’, and Z with ‘Socrates’. (More complex proofs are constructed by letting the conclusion of one proved deduction become a premise of another.)

Note that if ‘Socrates’ in the last line were substituted by the pronoun ‘He’, we would no longer be able to do the above matching (Z would have to be matched with both ‘Socrates’ and ‘He’). In order to prove the new conclusion we would need more rules. What is formally provable depends on the set of rules in our formal proof system.

The complexity of natural languages makes formal proving very difficult. Therefore propositions are usually expressed in *formal languages* that have very simple syntax and semantics. Below we have stated the deduction above in a *first order language*.

$\forall x : \text{man}(x) \rightarrow \text{mortal}(x)$   
 $\text{man}(\text{socrates})$   
 $\therefore \text{mortal}(\text{socrates})$

For simple formal languages with restricted expressive power there exist *complete* proof systems, i.e. any valid deduction expressible in the language is formally provable. The general rule is that the more expressive power a language has, the more difficult it becomes to find a complete proof system for it. First order languages represent in a way a maximum of expressive power combined with complete proof systems and have therefore been subject to much attention.

Figure 2

important is perhaps that we should aim for *verifiable* services. That is, to specify and design services in such a way as to make verification possible if requested. This should include both formal and informal verification techniques. If we shall be able to do some sort of formal verification we need to have formal specifications.

## 6 Language issues

Every description needs a language. Up to now we have argued for an object oriented approach to the description of services. Object orientation alone, however,

is not sufficient. We need to state what the objects do, i.e. provide their semantics, and for that a set of concepts must be defined in a precise “mathematical” way.

### 6.1 Mathematical descriptions

Mathematical texts are mostly written in a mixture of English (or some other natural language) and some formal notation. The descriptions rely on concepts with a precise mathematical meaning, such as sets, functions, mappings, etc. Proofs are not formal (see Figure 2), but are explications that

attempt to convey an understanding of the problems, constructions, solutions, etc., involved. Often there is a consensus that if someone cared to undertake the tedious task of translating them into formal proofs, this could be done.

Formal proofs, on the other hand, do not rely on any understanding of concepts, but consist of a series of string manipulations in conformance with given patterns (called rules or schemata). They are therefore particularly well suited to computers. A problem with mechanical theorem provers is that the search for a proof often leads to a “combinatorial explosion”, i.e. the number of cases that needs to be inspected grows to such proportions that the computer has to give up. The search for a proof must therefore be guided by heuristics or human interaction.

The advantages of formal proofs are as follows: First, they allow us to disregard human subjectivity. Second, one can look at mechanical theorem provers as tools that help humans in finding and verifying proofs and making the translation from informal to formal proofs less tedious. The third advantage is probably the most important for us. Programming languages are formal languages in the sense that they are “understood” by computers. When we write class definitions, we are thus using a formal language. Programs are best related to “mathematical concepts” if the concepts are described in a formal way too; this is even a prerequisite for formal program verification (see Section 5).

In order to provide formal descriptions, a formal language must be chosen. *Expressive power* is an important criterium. An outstanding class of languages are the *first order* languages used in first order logic (FOL). A large class of mathematical structures are definable in FOL, e.g. groups, rings, fields; see Figure 1. FOL is however limited in expressive power. For example, it is impossible to define the natural numbers  $\mathbb{N}$  or the real numbers  $\mathbb{R}$ . In order to single out  $\mathbb{N}$ , one has to move to higher order logic (HOL).

Efficient theorem proving is another important criterium. Theorems are important for several reasons. First, theorems supply us with explicating facts, thus improving our understanding of defined concepts. Second, if we are able to prove theorems, we are also able

to discover inconsistencies in our descriptions by proving a contradiction. Third, theorems may help finding and improving efficient implementations. The way graph theory already has been used to configure cost effective telecommunication networks clearly illustrates this fact.

The general rule is that the greater the expressive power of a language is, the less efficient theorem proving becomes. First order languages incorporate in a way a maximum of expressive power combined with completeness. Depending on the field of application, first order languages may be too weak in expressiveness, but also too strong. They may be too weak to define certain structures, as pointed out above, but too strong to allow for efficient theorem proving. The reason is that, although complete, first order logic is not *decidable*, i.e. proofs may be extremely long, and in general there is no way of telling whether a deduction is provable. (This fact is tightly related to the halting problem of Turing machines: There is no general way of telling whether a Turing machine with a given input tape will halt.) In order to obtain decidability one has to give up some of the expressive power of FOL. At the present time we (the authors of this article) have not gained enough experience to tell whether we should base our descriptions on another logic than FOL.

Developed methodologies, so-called *formal methods* or *formal description techniques*, make formal descriptions easier to write and understand. They can be based on FOL, HOL or others, or on some specific theory like set theory. Their convenience in use is, in addition to expressive power, an important criterium for an appropriate choice. Although based on some specific logic they can present to the user a richer language with e.g. possibilities for modularity, that is translatable (compilable) into a language of the logic they are based on, thus allowing for lucid and comprehensible descriptions. They can also be associated with computer applications that support automatic syntax checking, theorem proving, program verification, etc.

## 6.2 Object oriented descriptions

Object orientation is a design and description that does not only and necessarily imply the use of an object

oriented language. For a discussion of the principles of object orientation the reader is referred to a text book on the subject, e.g. (7). Some languages, however, encourage and facilitate working within this paradigm better than others. (Figure 1 illustrates object orientation by a small example written in Eiffel.) In addition to the typical characteristics of object oriented languages the following facilities are relevant for the description of services.

### *Facilities to handle concurrency and distribution*

At some level the problems that arise in connection with the fact that services involve the participation of several processes running in parallel and interconnected by a network have to be dealt with. A language that allows a smooth transition from higher level descriptions where distributional and concurrency aspects are transparent, to a lower level where these aspects can be handled, is preferred.

### *Facilities to handle real-time requirements*

The behaviour of certain services depends on elapsed time. “Call Forwarding on No Answer”, e.g., is triggered only when a call is not answered for say 20 seconds. Such time constraints must be expressible. One also needs to express time constraints in order to schedule processes running in a distributed environment.

## 6.3 Interface between objects and underlying concepts

Somehow we need to relate the objects to the concepts they rely on. Common practice is to describe this relation as comments in the code. For example, a procedure used to compute the factorial function will contain a comment stating this (usually presupposing that natural numbers and the factorial function are well-known, overlooking the fact that these too need to be defined). In Figure 1 the class *GROUP* is related to the mathematical concept *group* by the assertions, i.e. the comments as well as the *requires* and *ensures* clauses. The language used in the assertions contains elements belonging both to Eiffel (the object oriented language) and to the first order<sup>1)</sup> language used to define a group. For example, the *ensures* clause in the description of the *add* feature is an equation whose left hand side belongs to Eiffel and right hand side belongs to the



language used to define a group  $(G, o)$ . This mixed *interface language* is necessary to express relations between entities in the object world and entities in the mathematical world. The first order descriptions together with the interface language can be viewed as a “two tiered” specification of the object oriented descriptions, an approach that is developed by the Larch Project, see (4).

## 7 Conclusion

In order to make precise descriptions of services without going into all the implementational details, a set of concepts must be developed. By adopting an object oriented approach and supplementing object oriented descriptions with mathematically defined concepts it is possible to describe services on different levels of detail. In that way the requirements to descriptions coming from users, subscribers, and providers are met.

## References

- 1 Ehrig, H, Mahr, B. *Fundamentals of Algebraic Specification 2*. Berlin, Springer-Verlag, 1990. (EATCS Monographs on Theoretical Computer Science; volume 21).
- 2 Fraleigh, J B. *A First Course in Abstract Algebra*. Reading, Mass., Addison Wesley, 1976.
- 3 Guttag, J V, Horning, J J, Modet, A. *Report on the Larch shared language: Version 2.3*. Digital Equipment Corporation, Systems Research Center, 1990. (Technical Report TR 58.)
- 4 Guttag, J V, Horning, J J, Wing, J M. *Larch in five easy pieces*. Digital Equipment Corporation, Systems Research Center, 1985. (Technical Report TR 5.) Superseded by DEC SRC TR 58.
- 5 Hayes, I. *Specification Case Studies*. Englewood Cliffs, N-J., Prentice Hall, 1987.
- 6 Hoare, C A R. An axiomatic basis for computer programming. *Communications of the ACM*, 12, 576-580, 1969.
- 7 Meyer, B. *Object oriented Software Construction*. Englewood Cliffs, N-J., Prentice-Hall, 1988.
- 8 vanLeeuwen, J (editor). *Handbook of Theoretical Computer Science, volume B*. Amsterdam, Elsevier, 1990.
- 9 *CCITT Q.1200-Series of Recommendations*. Geneva, 1992.

---

1) In Figure 1 we have deviated a little from first order logic when writing the groups axioms in order to make them easier to read.

# Standardisation activities in the intelligent network area

BY ENDRE SKOLT

621.39.05:006  
681.324

## Abstract

*This paper gives an introduction to the ongoing standardisation activity on Intelligent Networks (IN). It contains an overview of the first available standard: Capability Set 1 (CS-1). The standard defines the service requirements for CS-1 and introduces the concept of service independent building blocks (SIBs). It also covers call modelling and functional architectural aspects.*

*The paper also gives an overview of preliminary plans for the next standardisation phase of IN, Capability Set 2.*

## Introduction

Standards in the Intelligent Network area are motivated by the interests of telecommunication service providers to meet existing and potential market needs for services in a rapid and cost effective manner. Another important motivating factor for IN standardisation is that the service providers seek to improve the quality of their service offer and to reduce the cost of network operations and management.

One of the objectives of Intelligent Networks (IN) is to allow the inclusion of additional capabilities to facilitate service and network implementation independent provisioning of services in a multi-vendor environment. That means to allow service providers to define their own services independently of equipment suppliers, and in the same way allow network operators to install switches and databases supplied by different manufacturers.

The target IN will be applicable to a wide variety of networks including Public Switched Telephone Networks (PSTN), Public Land mobile Networks (PLMN), Packet Switched Public Data Networks (PSPDN) and Integrated Services Digital Network (ISDN), both Narrowband ISDN (N-ISDN) and Broadband ISDN (B-ISDN).

## Capability Set 1 (CS-1)

CS-1 is the first stage of standardisation of the Intelligent Network as an architectural concept for the creation and provision of telecommunication services. It is, however, important to stress that CS-1 only contains a subset of IN capabilities. CS-1 can be viewed as follows:

- CS-1 is a subset of the target intelligent network architecture
- CS-1 provides network capabilities to support services defined by standardisation bodies. In addition CS-1 will support the introduction of services which neither are standardised nor part of the proposed set of targeted services. This enables service providers to offer tailor-made services to their customers
- CS-1 will provide a useful experience which can influence the future evolution of IN.

Traditionally, standardisation has been an activity which implied solutions based on existing technology. In the case of IN the standardisation has become closer to a research project where the results are advancing the current technology. However, due to the project size and the large number of participating organisations, compromises are necessary in order to

make progress, and the standards may therefore technically not represent the optimal solutions for each service provider.

CCITT SG XI started work on IN in 1989. There were different opinions concerning the time frame for which the group wanted to achieve implementable recommendations. Two opinions were confronted: On the one hand, strong market needs require implementable specifications in a very short time frame. It was also stressed that the market does not have time to wait several years for pilot trials.

On the other hand, it was argued that in order to achieve viable solutions it would be necessary to work on a long term basis.

As the work progressed it became clear that most of the participants favoured the first approach and aim for specifications already in 1992. (CS-1 will formally be approved in March 1993.) In Figure 1 some milestones of the work on IN is shown.

CCITT is a consultative committee which produces recommendations for the International Telecommunication Union (ITU). CCITT is organised into study groups, and they are open for all parties

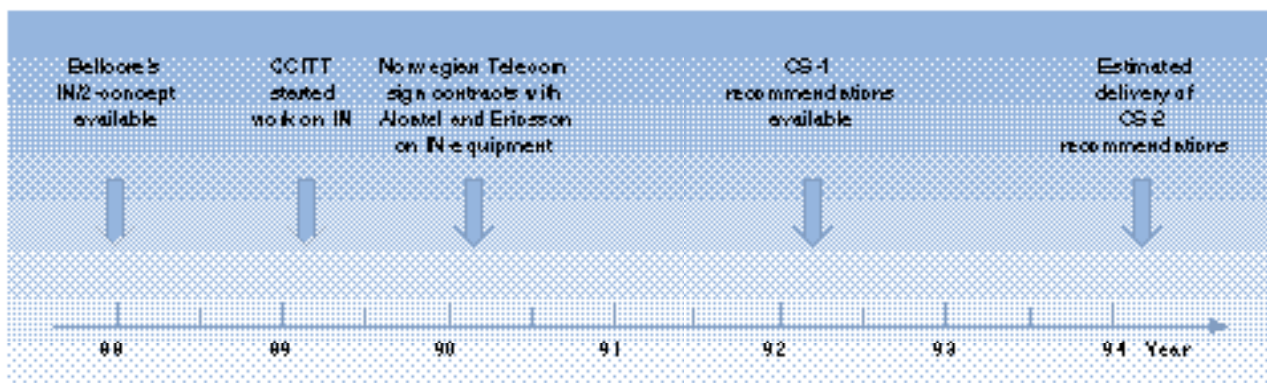


Figure 1 Milestones in the work on IN

in the telecommunication society. A number of equipment manufacturers, service providers, network operators and organisations representing the customers from all over the world have participated in the specification of CS-1.

The large number of participants imply that CS-1 is a compromise of many interests. There is, however, no doubt that Bellcore's early IN studies have had considerable influence on CS-1. In addition, the equipment manufacturers have been very active in order to prevent

the standards to impose too many changes to current switching technology.

## What is contained in Capability Set 1?

As mentioned above, CS-1 contains only a subset of the target IN architecture, but will give the telecommunication players a common platform for future evolution. In the first stage of the work much effort has been focused on the modelling of the IN concept and a conceptual model for IN has been developed to structure the concept. The model consists of four

planes where each plane represents a theoretical view of the services and network functions as follows:

- Service Plane. This plane illustrates the view of a service from a customer's point of view. A service is represented by means of generic blocks called service features.
- Global Functional Plane. This plane illustrates a service provider's view of the service creation process. The service and service features in the service plane are mapped to service independent building blocks (SIBs) in the global functional plane.
- Distributed Functional Plane. This plane illustrates a network operator's view of the network represented as functional entities<sup>1)</sup> and relations between them. The SIBs in the global functional plane will be mapped into functional entities in the distributed functional plane.
- Physical Plane. This plane models the physical network with different physical nodes and protocols.

CS-1 provides an IN platform, which is the base for provision of services. This platform has been developed on the basis of a set of benchmark services and service features. The services and service features of CS-1 are based on a set of Service Independent building Blocks (SIBs). The SIBs are the service components required for creation and composition of services.

An important accomplishment of CS-1 is the definition of communications protocols for the interface between the Service Switching Function (SSF) and the Service Control Function (SCF) (see definitions on page 49). CS-1 also contains other protocol standards. These will be discussed later in this paper.

The main results of CS-1 is in the area of service execution, that means the rules and procedures to establish, maintain and release a call. Service creation and service administration have not been dealt with in the same depth but may be a major working area in the next stage of IN standardisation (CS-2).

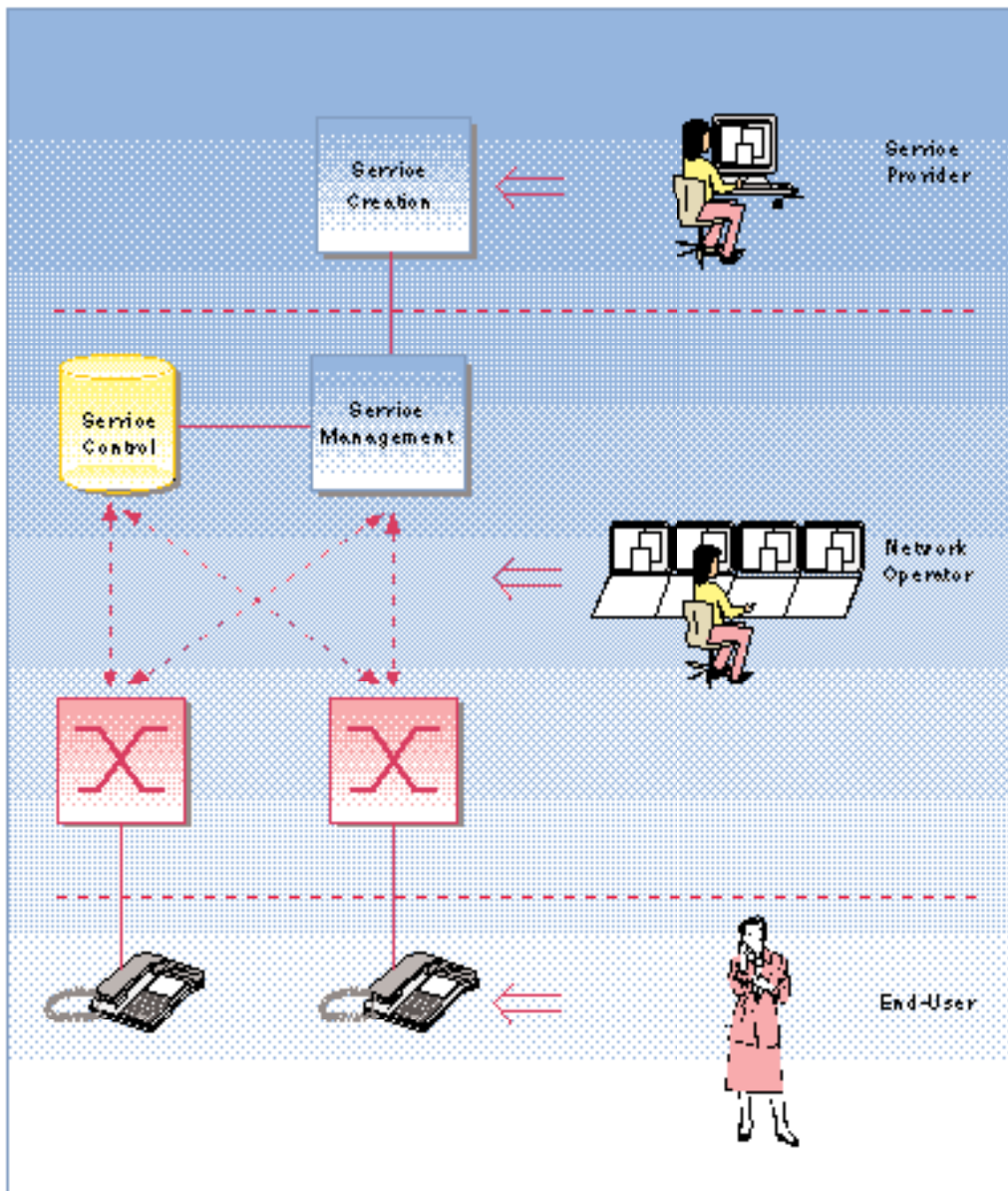


Figure 2 Parties in the telecommunication society

<sup>1)</sup>A functional entity is a grouping of functions distributed in the network.



## What will the tele-communication society gain from CS-1?

In this section I have summarised some benefits the different telecommunication parties (see Figure 2) will gain from CS-1:

The separation of switching functions, service logic, specialised resources and subscriber data enables the network operator to purchase equipment from different suppliers. Such equipment includes Service Control Points (SCPs), Service Switching Points (SSPs), Intelligent Peripherals (IPs) and Service Data Points (SDPs). In addition CS-1 will provide efficient network routing and give savings in network and transmission resources. However, the savings must be paid for by an increase in signalling traffic.

Flexible provision of services and the capability to offer services like Universal Personal Telecommunication (UPT) and Virtual Private Networks (VPN) across several networks operated by different service providers will also be possible. However, initially there will be several restrictions on utilising the full capabilities of these services.

One fundamental element of the IN concept is the service independent building blocks (SIBs). By combining SIBs the service providers and network operators may offer a large number of services. However, CS-1 only offers some guidelines for the introduction and deployment of services into the network.

End users and service subscribers will through advanced user-network interaction have access to a large variety of services including customised service profiles, flexible charging and user control.

The development cost in software and hardware for public network systems are considerable. Therefore, it is vital for the equipment manufacturers to agree on world wide standards in order to have large markets for their products. With CS-1 they have accomplished a global standard.

## CS-1 services

The service plane of the IN conceptual model illustrates the service from a user's perspective. A service may be composed of several service features or service components. A list of service features for CS-1 is shown in Table 1. See Q.1211 for a more comprehensive description.

The type of services which can be offered in CS-1 is classified as type A services. Services that require IN handling but cannot be supported by CS-1 are called type B. In CCITT recommendation Q.1211 the following definition is given:

“The CS-1 capabilities are intended to support service and service features that fall into the category of single ended, single point of control services referred to as type A, with all other services placed in a category called type B.”

Single-ended service features mean that these features only apply to one party in a call and are independent of any other parties that may be participating in the call. Single point of control means that a control relationship in a call can only exist to one Service Control Function at any point in time.

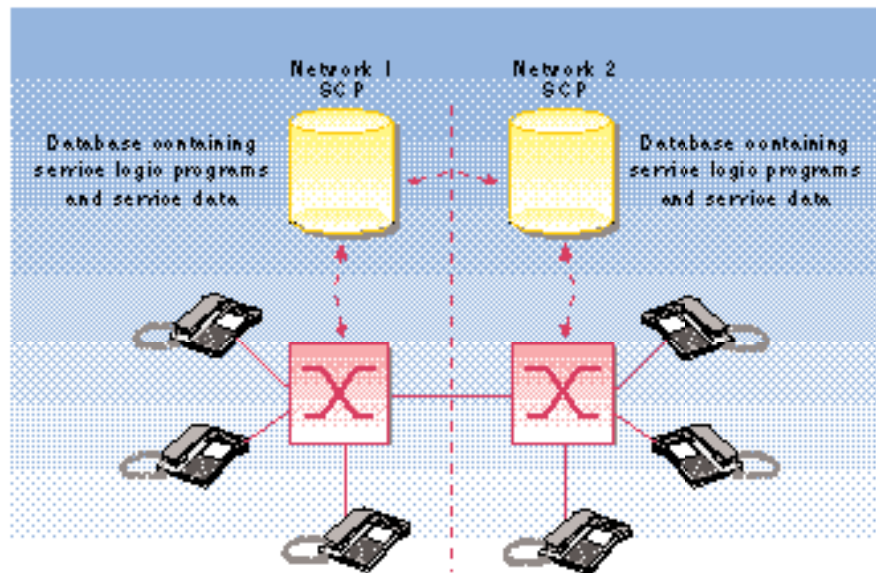


Figure 3 Participants in a conference call connected to switches in different networks

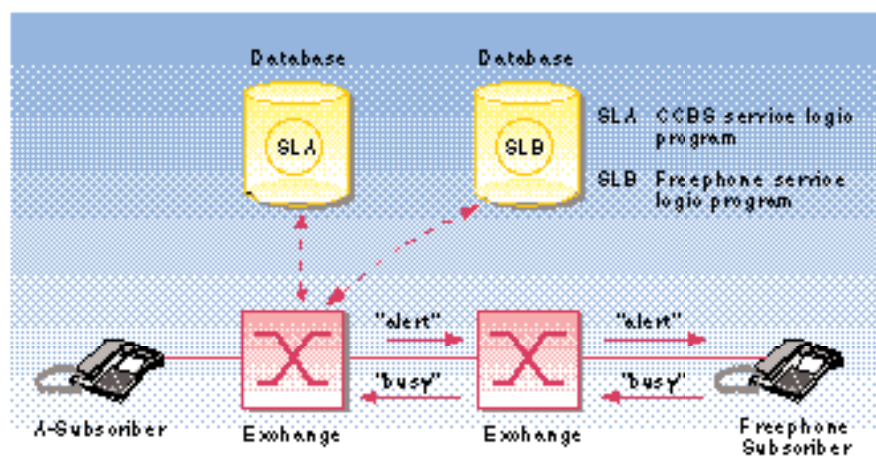


Figure 4 An example of feature interaction

## TRANSLATE

### Definition

Determines output information from input information

### Operation

This SIB translates input information and provide output information, based on the various input parameters. Parameters provided identify which file should be scanned for the translation. Translation can be based on either input information only, or on input information and the CLI.

For example, this SIB could be used for modifying input information (for instance dialled digits) into a standard numbering plan upon which network routing is based.

In conjunction with other SIBs, like Compare, the Translate SIB can provide the required functionality for time dependant routing.

### Potential Service Applications

Freephone, User-defined routing, VPN, UPT, Abbreviated dialling, Selective Call Forwarding on Busy/Don't Answer, Call Forwarding, Call Transfer.

### Input

Logical Start

Service Support Data

Type. Specifies the mode of operation for this SIB. Three modes have been identified: one number to one number, one number to more than one number, IA5 string to one number.

File Indicator. Specifies where the translation data file is located.

CIDFP-CLI. This CID Field Pointer Specifies the Calling Line Identification

CIDFP-Info. This CID Field Pointer specifies which call Instance Data is to be used as the Information

CIDFP-Translated. This CID Field Pointer specifies where in Call Instance Data that the translated data element is to be written

CID-Error. This CID Field Pointer specifies where in output Call Instance Data that the error cause will be written

Call Instance Data

CLI. Specifies the calling Line Identification

Information: Specifies the data to be translated

### Output

Logical End

Success

Error

Call Instance Data

Translated data. Specifies the data elements resulting from the translation

Error cause. Identifies the specific condition which caused an error during the operation of the SIB. The following

errorshave been identified for Translate: Invalid File Indicator, Invalid Information, Invalid CLI, Translation not available

### Graphical representation

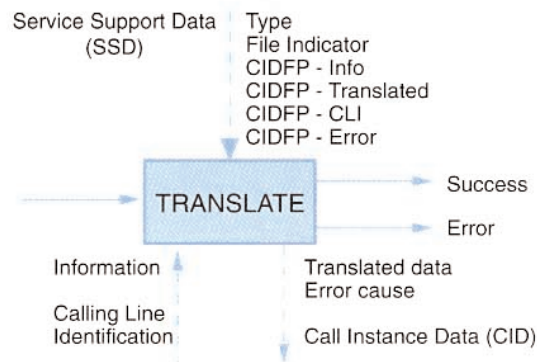


Figure 5 Stage 1 description of the Translate SIB



A conference call as a type A service will provide the participants very limited flexibility in the control and execution of the call. For example, the A-party will have to keep the control during the complete lifetime of the call and the associations of the parties physically take place in the switches controlled by only one Service Control Function.

For conference calls as type B the associations of the parties may physically take place in the switches controlled by more than one Service Control Function. It may also be offered that parties are added to or dropped off and that the control of the call can be transferred from one party to another. A conference call example where the involved parties are connected to switches in different networks and where service control interaction is necessary is shown in Figure 3.

In complex services the SCF will need rules to handle feature interaction<sup>2)</sup> between parties in a call. Since this aspect is not covered by CS-1, service providers and network operators need to rely on network and supplier specific solutions. The feature interaction problem is illustrated by an example where a Freephone service interacts with a Call Completion on Busy Subscriber (CCBS) service (see Figure 4). The calling user dials the Freephone number and a busy tone is returned. Then the calling user activates the CCBS service. In the network the following procedure will take place:

When the network receives the Freephone number, the called party's Freephone service logic program<sup>3)</sup> will be invoked. The service logic program will convert the Freephone number to a network routing number and the network will use this number to route the calls to the Freephone destination. When the Freephone user is busy the terminating exchange will return a busy signal. When receiving the busy

signal from the terminating exchange the calling user activates CCBS causing the network to start monitoring the called party's access. When the called party's access becomes free, the exchange will alert the originating exchange and the call will automatically be set up towards the called party.

In this case the two services will be independently executed since the Freephone service logic program belongs to the called party and the CCBS service logic program belongs to the calling user. The interaction problem occurs because the terminating exchange will view the second attempt as an ordinary call (not a Freephone call).

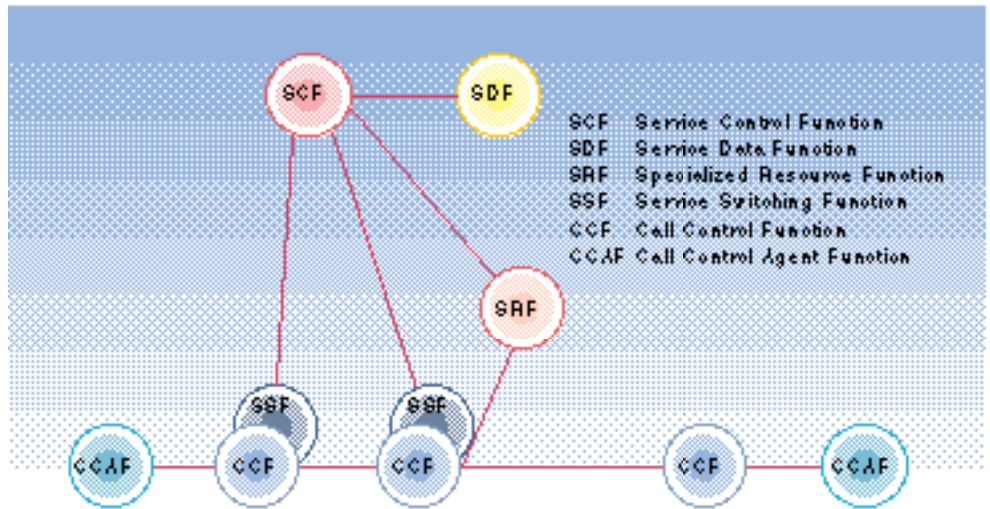


Figure 6 Call handling functional architecture

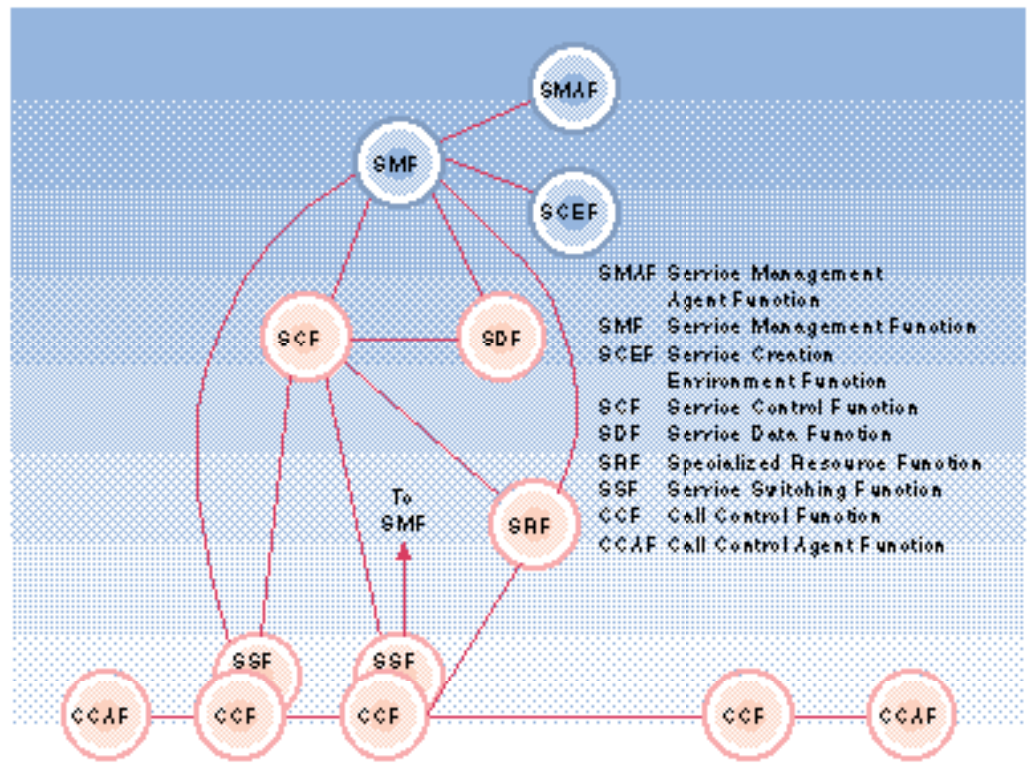


Figure 7 IN functional architecture including service management functional entities

<sup>2)</sup> A situation that occurs when an action of one feature affects an action or capability of another.

<sup>3)</sup> The service logic program is the software which is running when a service is executed.



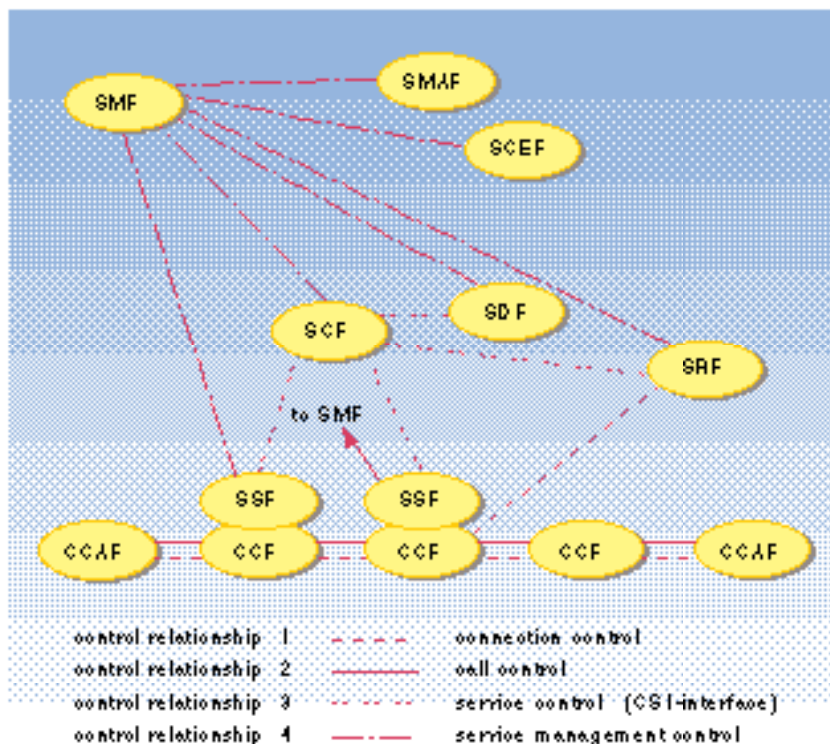


Figure 8 Control relationships for the IN functional architecture

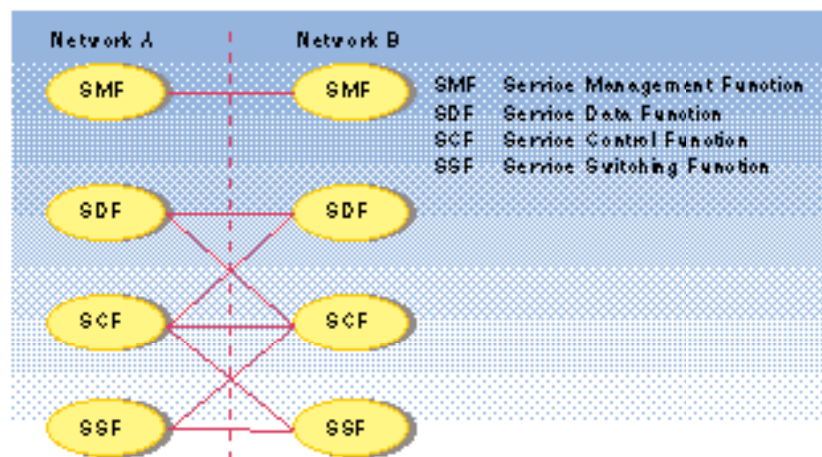


Figure 9 Network interworking relationships

There is a number of reasons why type B services are not included in CS-1. Type B services will require a complexity which has not been possible to explore within the time frame of CS-1. Type B services may also involve manipulation of abstract switching connections like legs<sup>4)</sup> and connection points. Existing switching equipment is not designed for such applications.

The switching manufacturers have been reluctant to introduce new software modelling in order to support these advanced capabilities.

On the other hand, type A services can be implemented by relatively simple control relationships between the switch and service logic database. In contrast the type

B services may require sharing of connection control and solutions based on distributed processing with large protocol complexity.

## Service creation

The global functional plane of the IN conceptual model is the service provider's view of the telecommunication network. This plane describes the service creation process by use of SIBs. For CS-1 fourteen SIBs have been defined, but rules and methodology to combine the SIBs have not been defined. A platform for creation and execution of services based on SIBs may be developed for CS-2. A list of all the SIBs defined for CS-1 is provided in Table 2.

SIBs are abstract representations of network capabilities that will exist in an IN structured network. As the name of these building blocks indicates, a SIB is independent of user services and the technology the services are developed from.

CS-1 contains stage 1 description, i.e. description of what each SIB does, and stage 2 description, i.e. a description of which functional entities the SIB involves and information flows between these functional entities. A stage 1 description of the Translate SIB is given in Figure 5.

## Functional architecture

The distributed functional plane contains the IN functional architecture and provides a network operator's view of the network. The functional architecture consists of functional entities (FE) where each FE represents a grouping of network functions, for example service control logic, specialised resources, service data, connection control, etc. The distributed functional plane also defines relationships between functional entities.

The following functional entities required for call handling are defined:

<sup>4)</sup> A leg is a theoretical representation of a connection path towards some addressable entity, e.g. an end user.

- **Call Control Function (CCF).** The CCF provides call/connection processing and control. It supports functions like trigger detection mechanisms in order to detect that a call requires IN handling and contains functions required to establish, manipulate and release call/connections.
- **Service Switching Function (SSF).** The SSF is closely associated with the CCF and provides the set of functions required for supporting interaction between the CCF and the Service Control Function (SCF).
- **Service Control Function (SCF).** The SCF contains the service logic program and the processing capability required to handle IN services.
- **Service Data Function (SDF).** The SDF contains customer and network data available for real time access by the SCF for execution of IN services. It interfaces and interacts with the SCF.
- **Specialised Resource Function (SRF).** The SRF provides the specialised resource such as digit receivers, announcements, conference bridges, etc. required for the execution of IN services.

The functional architecture is shown in Figure 6.

## Call control

The call control aspects involve the functional entities CCF, SSF and SCF. Some key principles for CS-1 call control are:

- The SSF to SCF relationship is service independent, i.e. CCF and SSF do not contain service logic specific to the services offered in CS-1.
- The CCF has the control of the connections in the switch at all times.
- In case of SCF failure the CCF/SSF should be able to proceed to a call completion or normal call release.
- The SSF should never have interaction with more than one SCF at a time for a single call.

## User-network interaction

In a large number of services there will be a need for user-network interaction in order to provide a user-friendly access. This interaction will be supported by a

variety of text messages (ISDN) and voice announcements. For this purpose the specialised resource function or an intelligent peripheral (physical network node) is used. An important principle in user-network interaction is that the Service Control Function will have the capability to suspend and resume service processing on behalf of the calling and called party.

With the enhanced capabilities for user-network interaction, user control will be offered to the customer in order to manage the services. Customer control includes the functionality to interrogate, modify and delete records in the customer's service profile. The customer control procedure primarily addresses the functional entities SSF, SCF, SDF and SRF. A customer control procedure will normally take place outside the context of a call.

## Service management

The CS-1 standards do not cover service management aspects, but the entities responsible for management have been identified:

- **Service Management Function (SMF).** The SMF may contain functions for deployment of service software and data to the network. Relationships with other functional entities have not been standardised.
- **Service Creation Environment Function (SCEF).** The SCEF may contain tools for creation and verification of services.
- **Service Management Agent Function (SMAF).** The SMAF may contain functions for access to the Service Management Function.

The functional architecture including the service management related functional entities is shown in Figure 7.

## Interfaces required for Capability Set 1

For CS-1 thirteen functional relationships or interfaces between functional entities have been identified. The functional relationships are shown in figure 8. In addition four types of control relationship have been defined in order to distinguish the type of physical interfaces. They are defined as follows:

- **Service control relationship.** This relationship involves the separation of switching functions from service control logic.
- **Connection control relationship.** This relationship contains capabilities to establish, supervise and release bearer connections (transmission paths).
- **Call control relationship.** This relationship contains capabilities to provide control between end-users for non-IN based services (e.g. ISDN supplementary services).
- **Management related control relationship.** This type of relationship is not standardised for CS-1.

For CS-1 the protocols between the Service Switching Function and the Service Control Function (SSF-SCF), the Service Control Function and the Specialised Resource Function (SCF-SRF) and the Service Control Function and the Service Data Function (SCF-SDF) are standardised. These protocols are all of the service control relationship type. With reference to the OSI layer model, they will be implemented as application layer protocols.

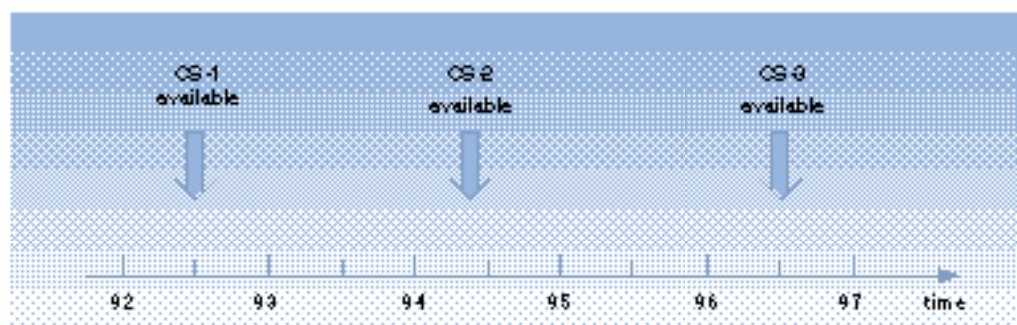


Figure 10 Preliminary dates for the issue of new capability sets

In addition to the work on interface standards for IN, CCITT has defined application layer structure for application protocols which is applicable to the IN protocols. For all the CS-1 specified protocols the same underlying protocols can be used, e.g. Transaction Capabilities Application Part (TCAP) or application protocols supporting remote operations.

## Network interworking

CS-1 has provided recommendations to be used within a single network in a multi-vendor environment. Therefore, the network operators will have to rely on bilateral or multilateral agreements in order to offer services across networks, e.g. Freephone services, Universal Personal Telecommunication services and Virtual Private Network services.

However, the interface between Service Control Function and Service Data Function may be implemented as an inter-network interface. An application of this interface is seen for the offering of credit calling services, where the Service Data Function may be a database which contains authentication data, e.g. personal identification number (PIN) codes in combination with account numbers.

The Service Control Function to Service Switching Function relationship is not considered as an inter-network interface. This interface would require additional security capabilities in order to be taken into account. Work on Open Network Provision (ONP) may have impact on whether or not this interface will be considered in future capability sets.

The interface between two Service Control Functions would violate the "single point of control" which is a requirement for the type of services to be offered within the scope of CS-1, thus not standardised. In the future evolution of IN this relationship must be considered, in particular in order to solve the service feature interaction problem.

The interface between two Service Data Functions is not standardised for CS-1. However, this relationship may be important in mobility applications for location handling and optimal distribution of data.

Reverse Charging	Attendant
Call Distribution	Mass Calling
Call Gapping	Split Charging
Call Limiter	Premium Charging
Call Queuing	Private Numbering Plan
O_Call Screening	One Number
T_Call Screening	Customised Ringing
Closed User Group	Call Logging
Customer Profile Management	Call Forwarding on BY/DA
Follow-Me Diversion	Call Forwarding
Origin Dependent Routing	Personal Numbering
Customised Recorded Announcement	Automatic Call Back*
Time Dependent Routing	Call Waiting*
Originating User Prompter	Multi-Way Calling
Abbreviated Dialling	Meet-Me Conference*
Authentication	Call Transfer*
Authorisation Code	Call Hold with Announcement*
Off Net Access	Consultation Calling*
Off Net Calling	

Table 1 CS-1 Service Features. Service features marked can only be offered with restrictions

<b>Algorithm:</b>	Applies a mathematical algorithm to input data to produce a data result.
<b>Charge:</b>	Determine special charging treatment for the call in addition to that normally performed by the basic call process.
<b>Compare:</b>	Performs a comparison of an identifier against a specified reference value.
<b>Distribution:</b>	Allows the user to distribute calls to different logical ends of the SIB dependent of user specified parameters.
<b>Limit:</b>	Limit the number of calls related to a service feature even though calls may not be causing congestion. Such limiting will be based on user specified parameters.
<b>Log Call Information:</b>	Log detailed information for each call into a file. The collected information may be used for management services, e.g. statistics.
<b>Queue:</b>	Provide sequencing of calls to be completed to a called party.
<b>Screen:</b>	Perform a comparison of an identifier against a list to determine whether the identifier has been found in the active list.
<b>Service Data Management:</b>	Enables the user's specific data to be replaced, added, changed, retrieved, incremented, decremented or deleted.
<b>Status Notification:</b>	Provide the capability to request for the status and/or status changes of network resources, e.g. subscriber line.
<b>Translate:</b>	Translate input data to output data, e.g. a Freephone number to a network routing number.
<b>User Interaction:</b>	Allows information to be exchanged between the network and the calling or called party.
<b>Verify:</b>	Provide confirmation that information received is consistent with the expected form of such information.
<b>Basic Call Process:</b>	For CS-1 the Basic Call Process has been defined as a specialised SIB which provides the capabilities for basic calls. See Q.1204 and Q.1214 for a more comprehensive description of the Basic Call Process.

Table 2 List of service independent building blocks for Capability Set 1



Interfaces between Service Management Functions have not been considered as an inter-network interface.

An illustration of the possible interfaces for network interworking is given in Figure 9.

## Future standardisation activity

It is a common view that it will take many years to accomplish the target IN which is characterised by attributes such as efficient use of network resources, subscriber control of service attributes and customisation of services, modularisation and reusability of network functions, integrated service creation and implementation, standardised management of service logic, etc.

The way that has been chosen to reach this goal is to define capability sets which are to be subject of standardisation activity. To ensure a smooth evolution towards the target IN the recommendations produced must fulfill two important criteria:

- Backward compatibility, that means that the next capability set must be an evolution of CS-1
- Open-endedness towards long term views.

At the moment there are no fixed plans for when the future capability sets will be ready. However, it has been proposed to issue results from the standardisation activity every second year. That would imply CS-2 recommendations in 1994-95, CS-3 recommendations in 1996-97, etc. (see Figure 10).

CS-1 is the first step towards the target IN. Even though the technical work has formally been completed there is a need for a maintenance activity of CS-1. This will include removal of inconsistencies in the current version as implementation experience is progressing. This work will be done in parallel with the initiation of the next capability set.

At this stage detailed planning of the next capability set, CS-2, has not yet started, however, it is likely that CS-2 will include studies in a number of areas such as:

- enhancement of service execution aspects
- service management including service creation
- service and feature interaction
- network interworking
- security aspects.

Modelling of the IN concept will be an important part of the work and may in addition to the above mentioned areas be influenced by the ongoing activities in broadband and mobile radio.

The following outlines some preliminary thoughts concerning the work items for CS-2:

### Service management

The future networks may be structured according to Intelligent Networks and Telecommunication Management Networks (TMN), where TMN is the concept for support of the network operator's and service provider's management requirements. This includes procedures for planning, provisioning, installing, operation and administration of telecommunication services. Since the IN concept also incorporates management aspects, some work has to be done in order to align the two concepts. The work has already started, and for CS-2 this will be a prioritised work item. Part of the work will include modelling of service management functional entities illustrated in Figure 7.

### Service interaction

One of the most complex areas of the work on IN is service interaction. Service interaction is understood as the mutual influence services may have on each other, and occurs when the execution of a service modifies the behaviour of the execution of another service. In this paper a service interaction example has been illustrated for a Freephone service and Call Completion to Busy Subscriber service, see Figure 4. In order to provide advanced services for CS-2, service and feature interaction must be studied in detail.

### Network interworking

The CS-1 recommendation set focuses on network internal aspects, and is very little concerned with network interworking

aspects. However, in order to accomplish standardised solutions for international services across several networks some work has to be done. For CS-2 it is proposed to initiate work in this area, both network interworking between public networks and public and private networks.

### Security

Security aspects will also have to be studied in CS-2. Basically there are two types which are important for IN:

- access control
- data control.

Access control contains the user identification, user authentication and user authorisation by the network, where the user may be end-users, service providers and network operators. Data control relates to the control of input data, which has been introduced by a user.

## References

- 1 *CCITT Recommendation Q.1204*, March 1992
- 2 *CCITT Recommendation Q.1211*, March 1992
- 3 *CCITT Recommendation Q.1213*, March 1992
- 4 *CCITT Recommendation Q.1214*, March 1992
- 5 *CCITT Recommendation Q.1219*, March 1992
- 6 Skolt, E. *Intelligente Nett: Kort Innføring*. Kjeller, Norwegian Telecom Research, 1991. (U1/91)

# IN control of a B-ISDN trial network

BY KIRSTI A LØVNES, FRANK BRUARØY, TOM HANDEGÅRD  
AND BENGT G JENSEN

681.324  
621.39.05

## Abstract

This paper describes and discusses the modelling of connection control functionality in a B-ISDN (Broadband ISDN) trial network under development at Norwegian Telecom Research. The main objective of our work is to experiment with IN control of B-ISDN. The virtual switch concept adopted from ETSI/NA6 enables a decoupling of service logic and network technology.

## 1 Introduction

So far the research and standardisation on IN and B-ISDN has progressed more or less independently. The basic IN concept, separation of service control and switch control, has been motivated by a need for rapid and independent evolution of services and technology and a need for advanced addressing facilities. At this point in time, IN architectures proposed by standardisation bodies (CCITT/ETSI) are not taking into account B-ISDN service requirements. On the other hand, effort has been put into the standardisation of a control architecture and signalling for B-ISDN without taking advantage of the results obtained from the work on IN.

B-ISDN is likely to offer a wide range of multimedia/multiconnection services (point-to-point and point-to-multipoint) needing complex call handling functions (8, 9). There is no longer a simple one to one relationship between a call and a connection. A call may be associated with a number of connections, the number and quality of the "active connections" varying as the call progresses. Furthermore, advanced call negotiation capabilities will be needed for interworking between end-users with different terminal capabilities. An evaluation of the B-ISDN service requirements leads to a control architecture which enables calls to be established and con-

trolled independently of associated connections. This implies separation of service handling and connection handling functionality.

A strong relationship exists between the basic IN concept and the concept of service and connection control separation in the B-ISDN control architecture. However, the IN architectures proposed by standardisation bodies will need some modifications to incorporate B-ISDN requirements. Taking both the basic IN concept and B-ISDN service requirements into account, we want to visualise the separation between service processing functionality and functionality associated with manipulation of the communication resources in the underlying B-ISDN network. Our implementation of the interface between service logic and network technology is based on the Connection Control Socket described in the IN Connection Control Model proposed by ETSI/NA6 (1). This paper describes and discusses a possible implementation of the model in a B-ISDN network.

## 2 Overview of the B-ISDN trial network

Norwegian Telecom Research participates in a wide range of B-ISDN activities. As part of these activities, a B-ISDN trial network is being developed. The network will serve as a testbed for IN supported B-ISDN connection control functionality and ATM (asynchronous transfer mode) traffic and performance.

### 2.1 What is ATM and B-ISDN?

B-ISDN is the new broadband network currently under development and standardisation by CCITT (6, 7). It offers the prospect of extending the range of services offered by ISDN considerably. B-ISDN is planned to support services with both constant and variable bit rate for transmission of data, voice, sound, still and moving pictures and multi-media applications which may combine any of these components.

The transfer mode chosen for B-ISDN is called *asynchronous transfer mode* (ATM). The term *transfer* comprises both transmission and switching aspects; thus ATM is a technique for transmitting and switching information in a network.

All information to be transferred in ATM is contained in *cells* of a fixed size. The cells have a 48 octet information field and a 5 octet header. The information field is available for the user. The header carries information which is used by the ATM network itself. Its main component is a label for identification of cells.

ATM is a connection-oriented technique. Individual connections are recognised from the label field inside the ATM cell header. The term *asynchronous* refers to the fact that cells belonging to a connection are sent according to the actual need, and thus may appear in an irregular way. This is illustrated in Figure 2.

The connections offered in an ATM network are called *virtual channel connections*. A virtual channel connection is a concatenation of one or more *virtual channel links*. A virtual channel link typically spans a single physical link between two network nodes (switches, terminals, etc). Each virtual channel link is assigned an identifier. This identifier is called the *virtual channel identifier* (VCI) and is part of the cell header. The VCIs for a connection normally remain unchanged for the duration of the connection, although there may be exceptions, as explained later.

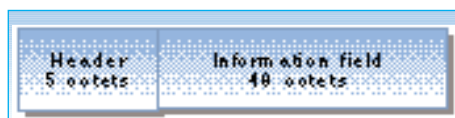


Figure 1 ATM cell

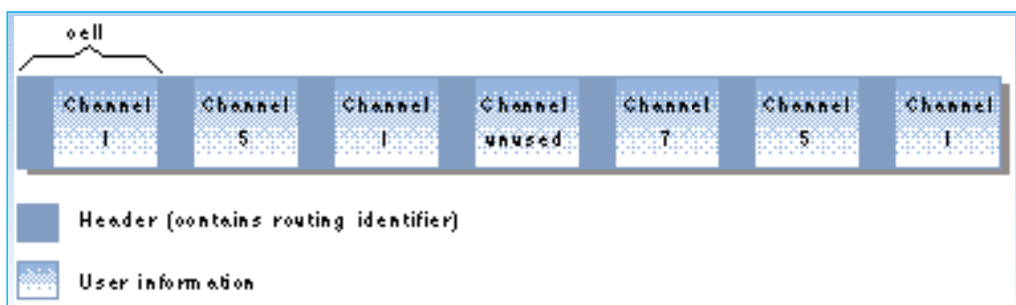


Figure 2 ATM principle

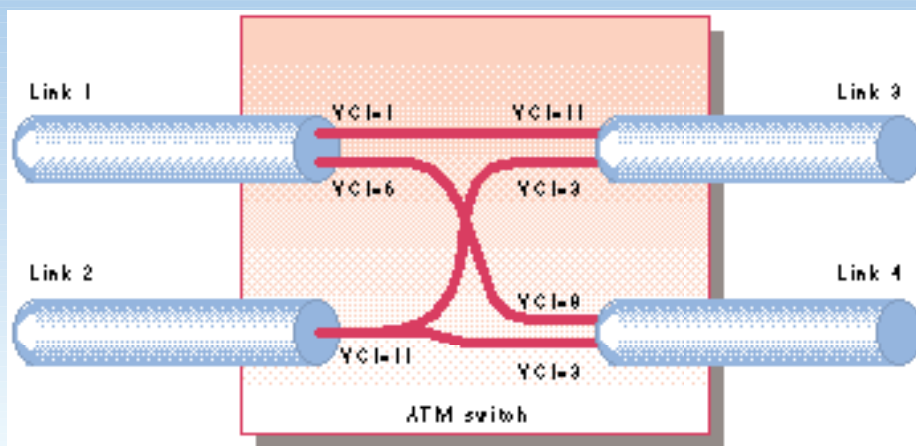


Figure 3 ATM switch

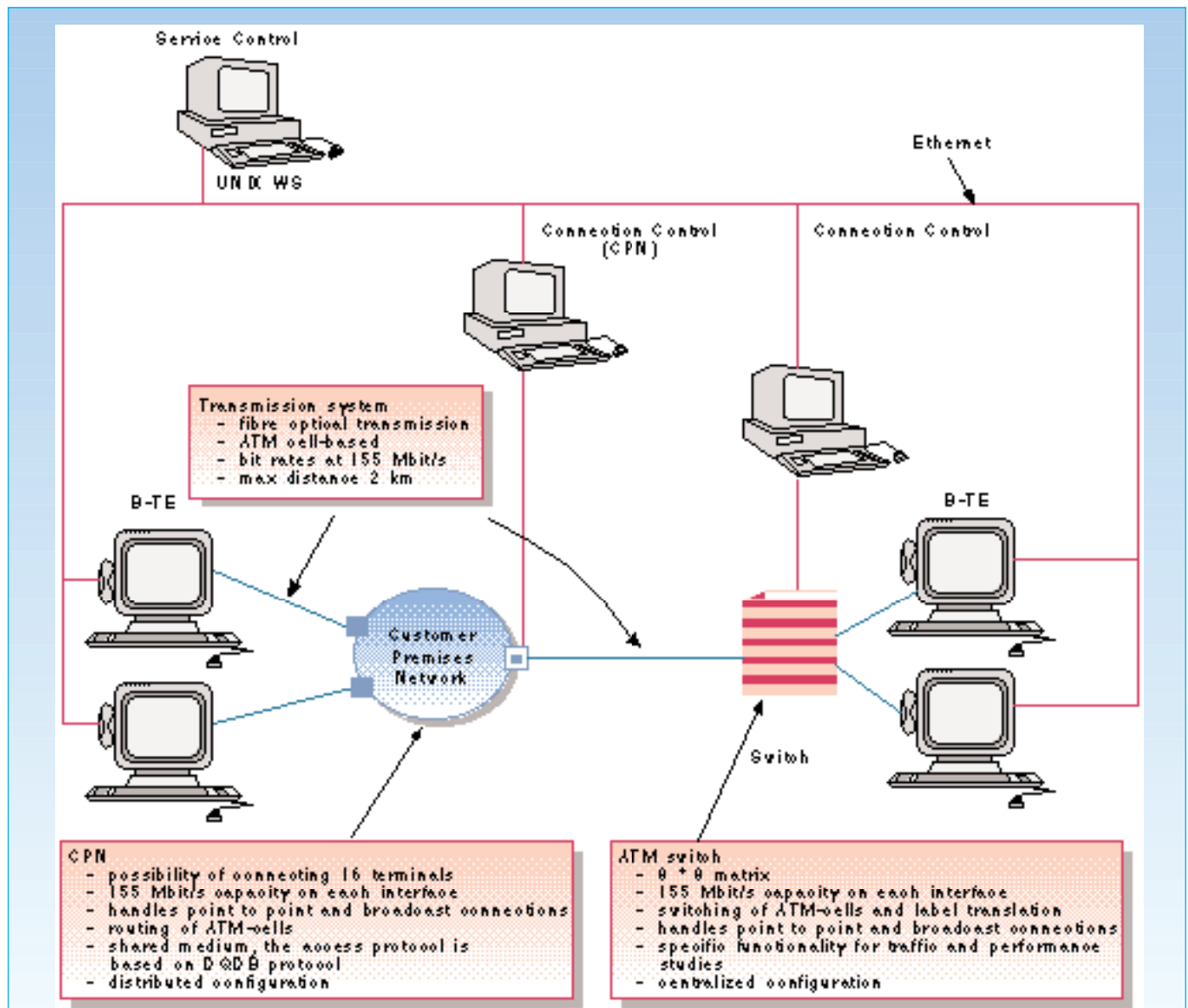


Figure 4 The B-ISDN trial network



The virtual channels are unidirectional. All cells associated with an individual virtual channel connection are transported along the same route through the network. Cell sequence is preserved, but cells may be lost. A bidirectional connection is realised by means of two unidirectional virtual channel connections.

The ATM switch is capable of interconnecting incoming virtual channel links with outgoing virtual channel links. This implies two main tasks: Translation of VCI and cell transport from ingoing to outgoing physical links.

Point-to-point, point-to-multipoint as well as multipoint-to-point configurations are possible. Figure 3 illustrates point-to-point and point-to-multipoint configurations. As shown, point-to-multipoint configurations imply copying of cells from an incoming physical link (link2(vci1)) to two or more outgoing physical links (link3(vci3) and link4(vci3)). On the contrary, multipoint-to-point configurations imply multiplexing of cells from two or more incoming virtual channel links to one outgoing virtual channel link.

The switch is capable of differentiating between reserved and through-connected connections. For a connection which is

only reserved, cell transport on the given connection is inhibited although resources to carry out the cell transport for the connection have been reserved.

## 2.2 Basic components of the B-ISDN trial network

Figure 4 shows an early version of the trial network consisting of an ATM switch, a customer premises network (CPN), terminals, transmission system and control units (service control units and connection control units). The network elements are interconnected by ATM based optical transmission systems with bit rates of 155 Mbit/s. Later versions of the trial network will be expanded with more ATM network elements (switches, CPNs, terminals, etc.). Furthermore, interworking units enabling communication with MANs, LANs and ISDN will be introduced.

As seen from Figure 4, the optical links are used for user data only (i.e. speech, audio, video). Control information is sent on a separate signalling network implemented by standard Ethernet. The use of Ethernet as a dedicated signalling network enables a decoupling of signalling and transport problems.

## 2.3 Introduction to the B-LAB connection control implementation

### 2.3.1 Overall control architecture

The connection control activity is a cooperation between IN and broadband research activities. The main objectives of our work is to verify system concepts and architecture, implement a connection control system for the trial network and explore the interaction between services (existing and future) and the ATM network. Through our modelling and implementation work we also gain experience with object oriented techniques.

As shown in Figures 4 and 5, the control architecture is composed of two control domains: service control (SC) and connection control (CC). In Figure 5 one CC node controls its separate ATM switch. Arrows are indicating the interfaces between a CC node and the SC domain, between two CC nodes, between a CC node and a terminal, between the SC domain and a terminal and between a CC node and an associated ATM switch.

In summary, the service control domain contains the service logic necessary to process service requests from end-users, while the connection control domain contains the functionality needed to establish, modify and release the requested connections in an ATM network. A more detailed description of the functionality of SC and CC follows in section 3.

### 2.3.2 The SC-CC interface

As our implementation of the SC-CC interface is based on the Connection Control Socket Model (CCSM) developed by ETSI/NA6, a brief description of its basic objects and service primitives will be given before going into a more detailed discussion of our control system.

The Connection Control Socket Model is one of the components of the Connection Control Model (CCM) in the ETSI IN architecture (1). The other two components, Basic Call Model and Connection Segment Relationship Model, are not included in our prototype.

The Connection Control Socket Model was chosen for our implementation as it seems to be in line with some of the main IN objectives:

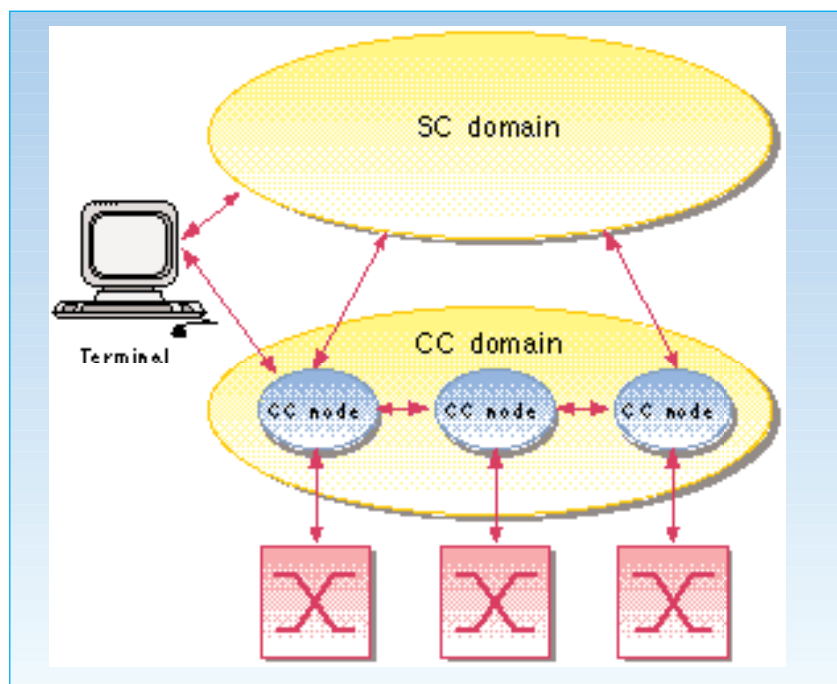


Figure 5 Overall control architecture

- The model allows a network independent execution of services.
- The model provides the services (call instances) with a simple consistent view of the underlying network.
- The model is powerful in terms of providing a large variety of connection configurations.

In this model the service instance views the network as a virtual switch object consisting of two types of attributes: legs and connection points. Each virtual switch may own several legs and connection points.

A leg is an abstract representation of a communication path extending from a connection point to an addressable network entity (e.g. a terminal). A leg may be unidirectional or bidirectional.

A connection point (CP) represents an interconnection allowing information to flow between legs. The legs associated with a connection point may be joined (connected to the CP) or split (disconnected from the CP). In our implementation of the socket, a leg is always associated with a connection point.

Figure 6 illustrates a socket containing a connection point and three associated legs. Two of the legs are joined (i.e. information flows between them), while the third leg is split from the connection point (i.e. no information is allowed to flow between this leg and the other two legs).

Below follows a short description of the service primitives across the SC-CC interface:

*open\_socket*: This message is sent from SC to CC to indicate the start of a session.

*create\_cp*: This message is sent from SC to CC to request creation of a connection point. In our implementation a connection point must always be created before legs are created.

*create\_leg*: This message is sent from SC to CC to request creation of a leg. *Create\_leg* implies reservation of necessary communication resources in the ATM network.

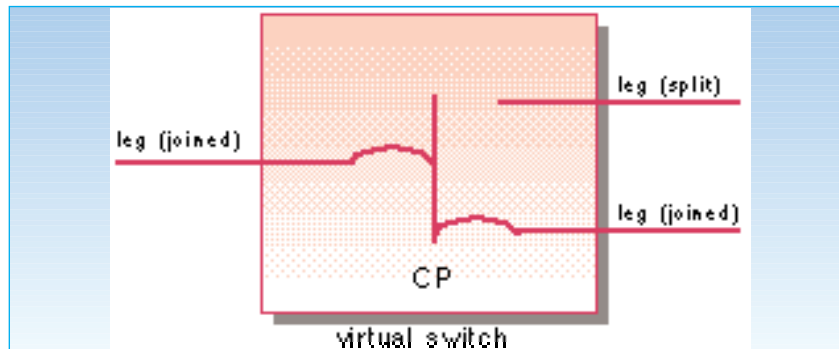


Figure 6 A socket containing a connection point and three associated legs

*join*: This message is sent from SC to CC to request a leg to be joined to a connection point. *Join* implies through-connect of reserved communication resources in the ATM network.

*modify\_leg*: This message is sent from SC to CC to request modification of QoS for a leg.

*split*: This message is sent from SC to CC to request a leg to be split from a connection point. *Split* implies disconnect of associated communication resources.

*free\_leg*: This message is sent from SC to CC to request a leg and all associated resources to be released. *Free\_leg* implies release of associated communication resources.

*free\_cp*: This message is sent from SC to CC to request a connection point and all associated resources to be released.

*close\_socket*: This message is sent from SC to CC to indicate the termination of a session.

Due to limited signal multiplexing capability in the ATM trial network, some restrictions are put on the possible combinations of legs and connection points:

- The legs associated with a connection point must be all unidirectional or bidirectional.
- For the unidirectional case, at most one incoming leg may be joined at a time (no restrictions on number of joined outgoing legs), i.e. effectively supporting point-to-multipoint configurations.

- For the bidirectional case, at most two legs may be joined at a time, i.e. effectively supporting point-to-point configurations.

### 3 Functionality of service control and connection control

This section gives a more detailed description of the overall control architecture introduced in chapter 2. A service example is used to illustrate the role and functionality of the two control domains.

#### 3.1 A service example

B-ISDN is likely to extend the range of services offered by ISDN considerably. The ATM network provides flexible bandwidth allocation mechanisms enabling a variety of multiconnection/multiparty services. To offer these services, the following call handling capabilities seem to be necessary:

- call (context) negotiation and terminal compatibility check
- establishment of multiconnection calls
- establishment of multiparty calls (e.g. broadcast and conference calls)
- modifications of Quality of Service for an established connection
- dynamic modifications of number of "active connections" in a call (i.e. the number of established connections may vary during the lifetime of a call)
- dynamic modification of number of active parties in a call (i.e. the number of communicating parties may vary during the lifetime of a call).

Below follows a service scenario including these call handling capabilities. The scenario involves Rosemary, her little baby, doctor Smith (the family doctor) and doctor Jones (a dermatologist).

*Rosemary's baby has been crying all night, and Rosemary suspects that it has something to do with the red spots appearing several places on her skin. Maybe it is some kind of itchy skin eruption? Rosemary decides to call a doctor to hear his opinion. She carries the baby to the videophone, and calls doctor Smith, the family doctor. "It's a nice thing this videophone", she thinks. "It's much easier to just show the problem to the doctor instead of trying to explain it. Now, maybe doctor Smith can order some treatment directly?"*

*Doctor Smith, however, feels uncertain. "There are so many kinds of skin problems these days, it's impossible for me to keep up", he says, "I'd better call a specialist. Don't hang up!" Doctor Smith temporarily parks the call to Jane and makes a voice call to doctor Jones. To keep the communication costs as low as possible, doctor Smith always tries to avoid using picture when it is not really necessary. After explaining the problem to doctor Jones, doctor Smith returns to Rosemary. "I shall connect you to doctor Jones. He is much better skilled to take care of your problem."*

*When doctor Jones gets the picture of Rosemary's baby on the screen, the first thing he does is to press a button that gives him an improved quality video picture. "It's a good thing you have a modern videophone that can supply high*

*quality video", he tells Rosemary. "With only standard quality video, it's not possible to tell one skin eruption from the other." Doctor Jones decides that Rosemary can treat the baby's skin herself, using healing ointment and bactericidal baths. Doctor Jones calls the new medical video database. He knows he can get a video clip there, illustrating how the treatment is to be done. He identifies the video clip, and orders it to be displayed both to himself and to Rosemary, giving him the opportunity to supplement the illustrated treatment.*

### 3.2 Service control

Assume doctor Smith has a UPT (universal personal telecommunication) subscription. Rosemary dials his UPT number (logical address), SC starts processing the service request, detects the address to be a UPT address and checks a database to find the physical address of doctor Smith's present location. SC now knows where to place the call.

Rosemary has requested a videophone call (standard quality video and speech) to doctor Smith. SC therefore checks if doctor Smith has the requested terminal capabilities at his present location and is free to accept the call. As part of this phase, SC negotiates the required quality of service (QoS) between the two end-user terminals. If the terminals are equipped with different video codec standards, it may be necessary for SC to involve an interworking unit in the network capable of handling the translation from one video codec standard to the other. In our example we assume the two terminals to be compatible, thus no interworking unit is needed.

Note that number translation and Quality of Service negotiation are performed by SC. CC gets involved (a connection control socket is opened) first after knowing the physical location and terminal capabilities of doctor Smith. At this point SC requests CC to create two connection points for the call. Next, SC requests creation of two sets of bidirectional point-to-point legs; one set for speech and one set for video. At this stage in the call, the communication paths from Rosemary to doctor Smith are reserved, but not through-connected. This implies that information is not allowed to flow between the two end-users yet.

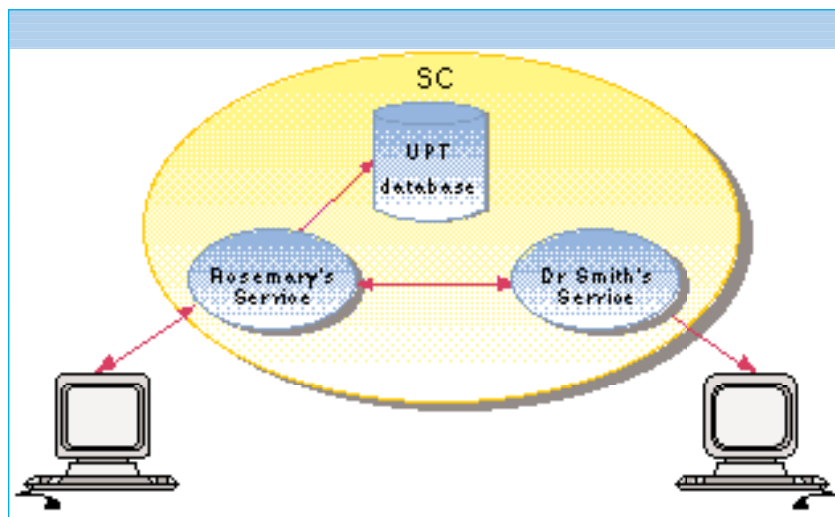


Figure 7 Localisation of called party

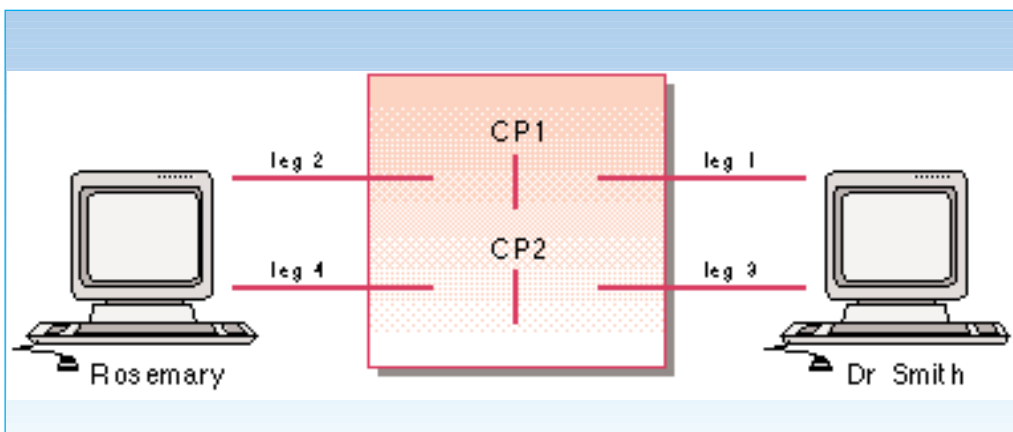


Figure 8 The reserved speech and video connections (Rosemary and doctor Smith)



The following sequence of requests are used for reserving the speech and video part:

```
open_socket
create_cp (cp1)
create_leg (cp1, leg1, speech, bidirectional, Dr Smith)
create_leg (cp1, leg2, speech, bidirectional, Rosemary)
create_cp (cp2)
create_leg (cp2, leg3, video, bidirectional, Dr Smith)
create_leg (cp2, leg4, video, bidirectional, Rosemary)
```

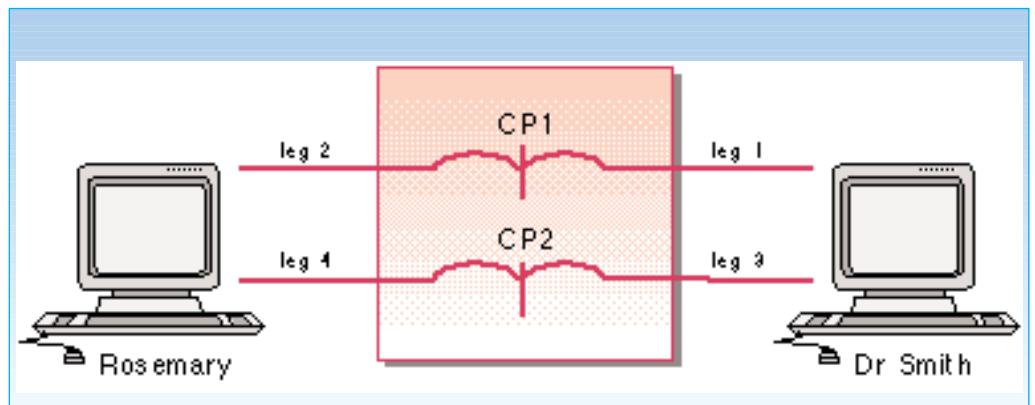


Figure 9 The through-connected speech and video connections (Rosemary and doctor Smith)

Rosemary receives an alerting indicating that the phone is ringing at doctor Smith's. Detecting that doctor Smith picks up the phone, SC requests each pair of legs to be joined to their associated connection points. The following sequence of messages is sent from SC to CC to through-connect the speech and video part:

```
join (leg1)
join (leg2)
join (leg3)
join (leg4)
```

This causes the call to be through-connected and the two persons may communicate with each other via their videophone terminals. Figure 9 shows the through-connected speech and video connections.

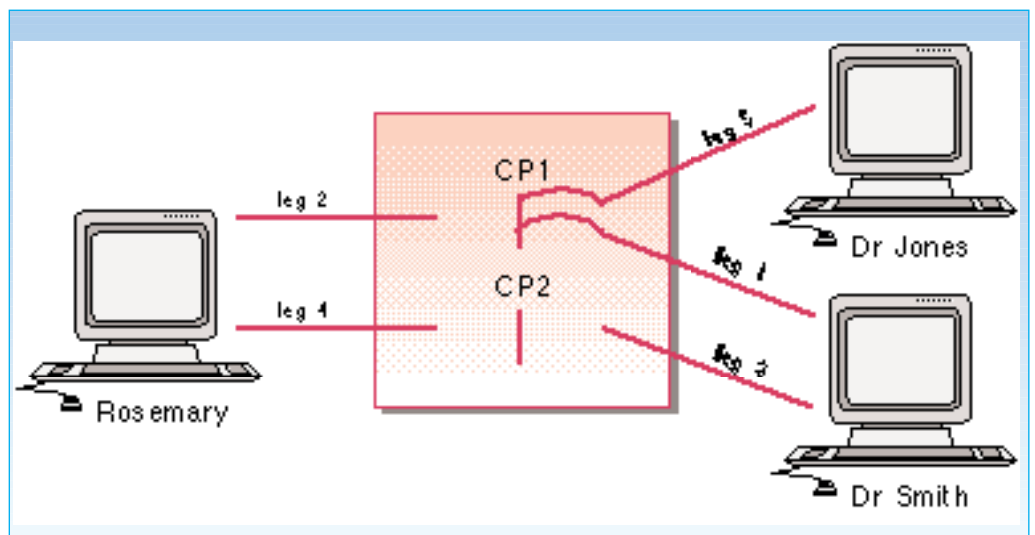


Figure 10 Call hold (Rosemary)/call transfer (doctor Jones)

Later, doctor Smith puts the call from Rosemary on hold and calls the dermatologist, doctor Jones. As a consequence, SC tells CC to split the speech and video legs from Rosemary and the video leg from doctor Smith and to set up a new speech leg towards doctor Jones. The following messages are sent from SC to CC:

```
split (leg2)
split (leg4)
split (leg3)
create_leg (cp1, leg5, speech, bidirectional, Dr Jones)
join (leg5)
```

Although the connections between Rosemary and doctor Smith are disconnected, the call is not released and the network resources are still reserved. Figure 10 shows the present situation.

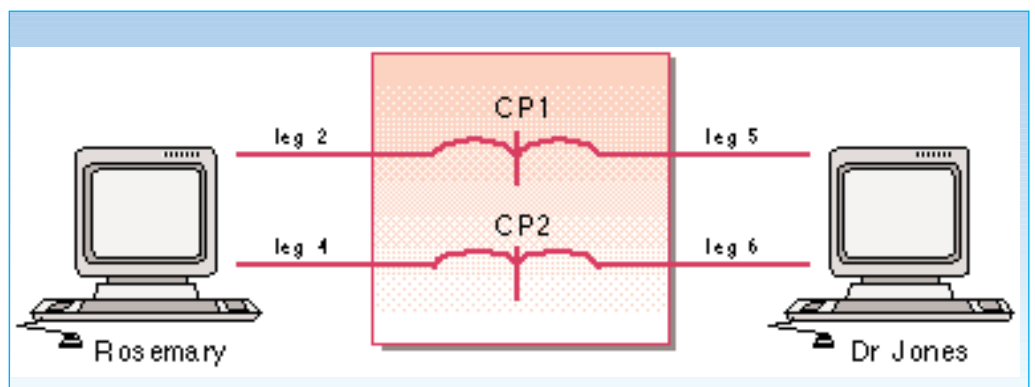


Figure 11 The through-connected speech and video connections (Rosemary and doctor Jones)

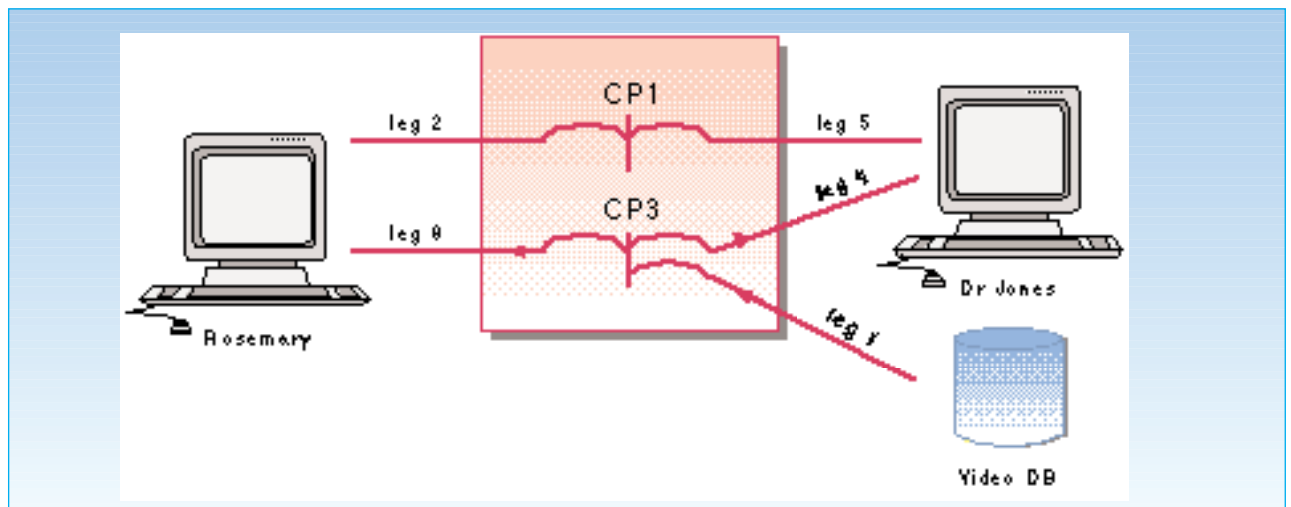


Figure 12 Three-party configuration including video distribution

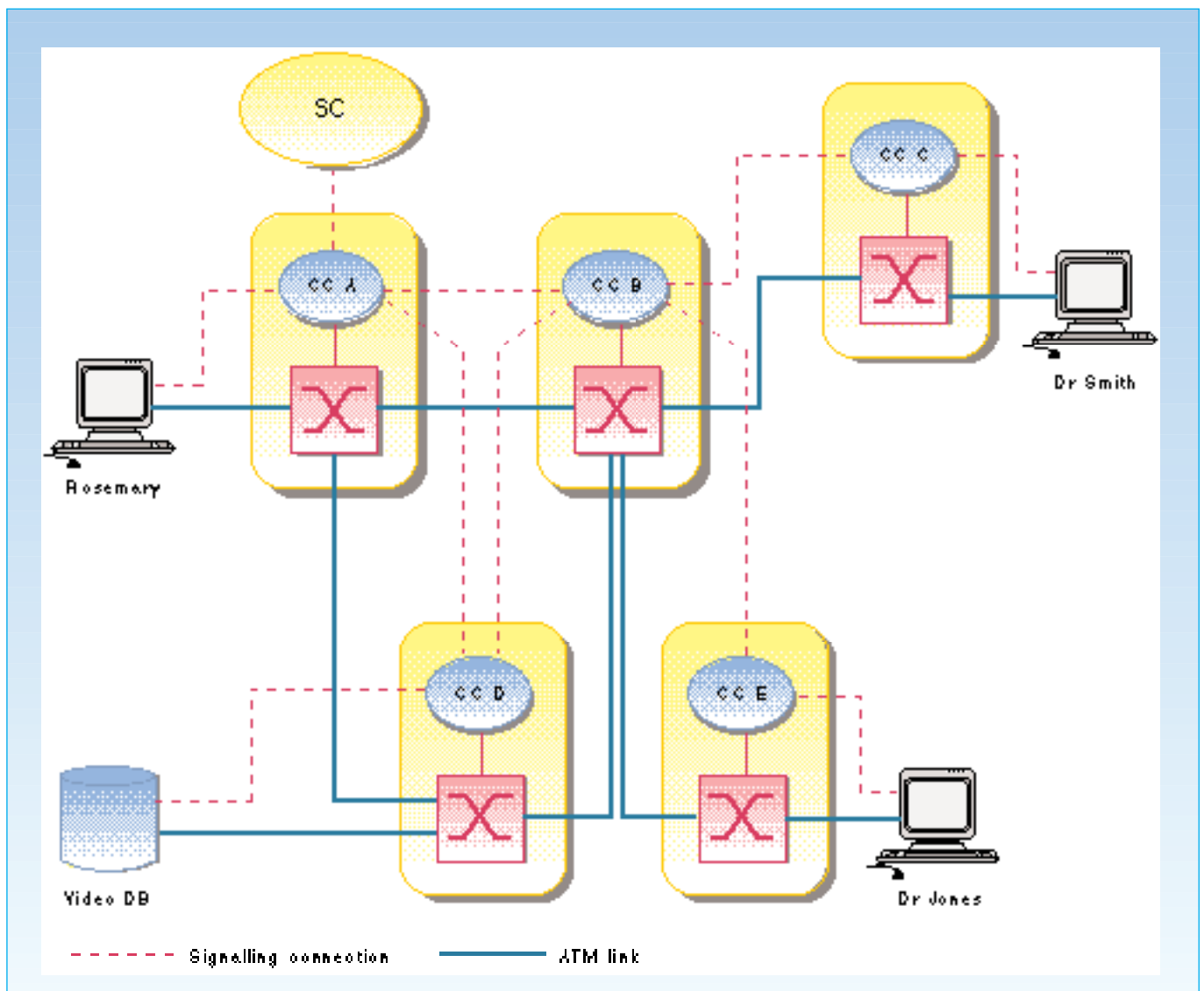


Figure 13 Example network

After having consulted doctor Jones, doctor Smith returns to Rosemary. He says goodbye and transfers her call to doctor Jones. As a result, SC requests all legs to doctor Smith to be released. The following sequence of messages is sent from SC:

```
split (leg5) #Doctor Smith returns to
# Rosemary
join (leg2)
free_leg (leg1) #Rosemary is transferred
# to doctor Jones
free_leg (leg3)
create_leg (cp2, leg6, video, bidirectional, Dr Jones)
```

```
join (leg4)
join (leg5)
join (leg6)
```

Doctor Jones realises that he needs a much better picture quality to be able to make a thorough examination of the baby's skin eruption. This calls for a renegotiation of the quality of service between the two end-users for the video part of the communication. If both terminals are capable of supporting high quality video, the following messages are sent from SC to CC:

```
modify_qos (leg4, video_high)
modify_qos (leg6, video_high)
```

video database and orders the video clip demonstrating the treatment to be presented to Rosemary and himself. The video part of the communication between Rosemary and doctor Jones is now released and a new unidirectional point-to-multipoint communication path is set up from the video database to Rosemary and doctor Jones. In order to do this, the following sequence of messages are sent from SC to CC:

```
free_leg (leg4)
free_leg (leg6)
free_cp (cp2)
create_cp (cp3)
create_leg (cp3, leg7, video, upstream, VideoDB)
```

After having identified the type of skin eruption, doctor Jones calls the medical

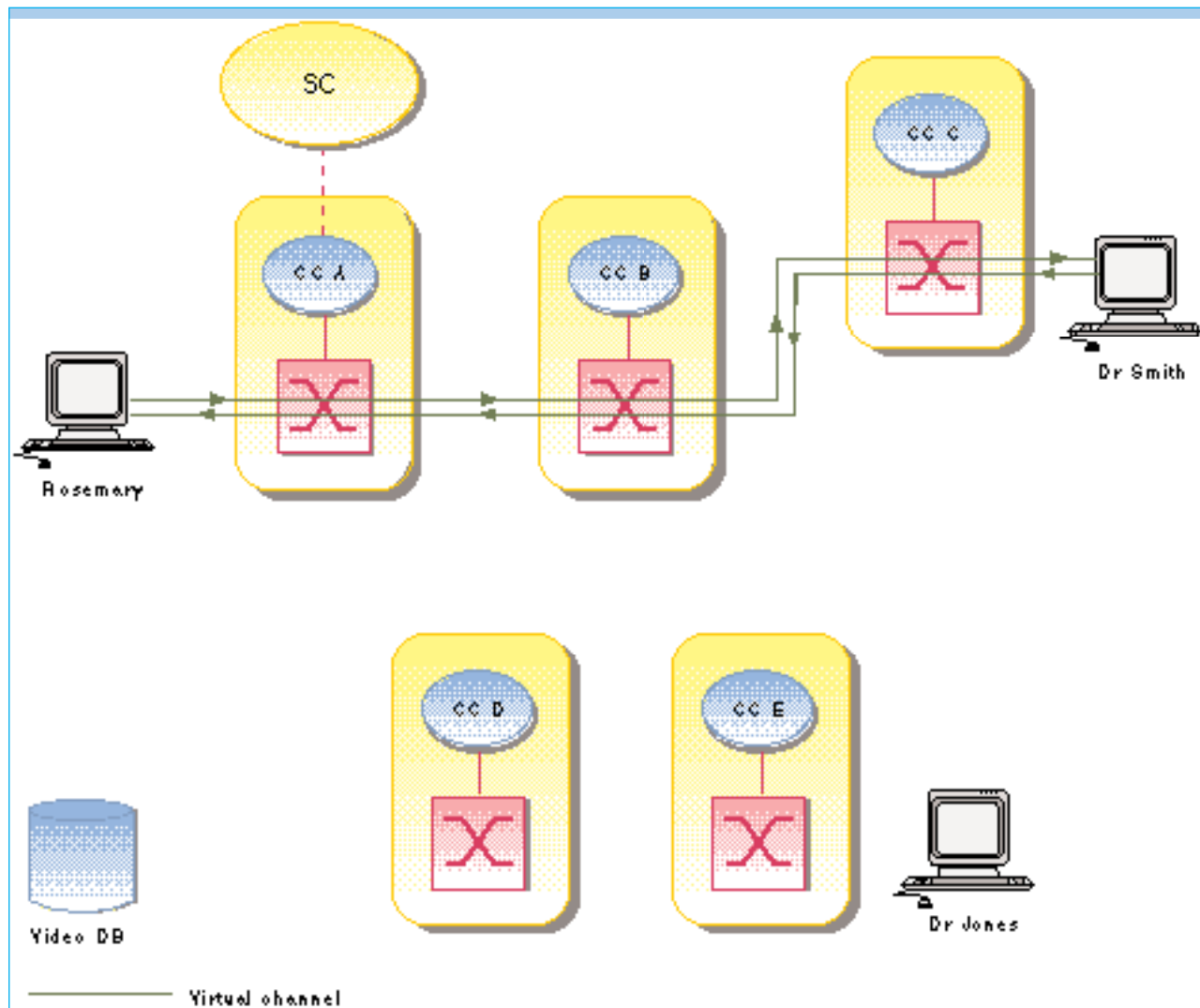


Figure 14 Two-party configuration



*create\_leg (cp3, leg8, video, downstream, Rosemary)*

*create\_leg (cp3, leg9, video, downstream, Dr Jones)*

*join (leg7)*

*join (leg8)*

*join (leg9)*

The speech part between Rosemary and doctor Jones is still through-connected in order for doctor Jones to explain the different steps of the treatment. Figure 12 shows the three-party configuration with video distribution.

### 3.3 Connection control

The task of CC is to set up, release and manipulate connections in the ATM network, making as efficient use of the network resources as possible. To show what this implies, we will first look in some detail at how the first speech connection is set up, and then extend the picture by looking at a couple of three party configurations. We assume the network configuration shown in Figure 13. Note that in this example, SC interacts with the CC node running in Rosemary's local exchange (exchange A).

#### 3.3.1 Two-party configuration

The sequence of requests needed from SC to set up the initial speech connection was given in section 3.2 and is repeated below.

*create\_cp (cp1)*

*create\_leg (cp1, leg1, speech, bidirectional, Dr Smith)*

*create\_leg (cp1, leg2, speech, bidirectional, Rosemary)*

*join (leg1)*

*join (leg2)*

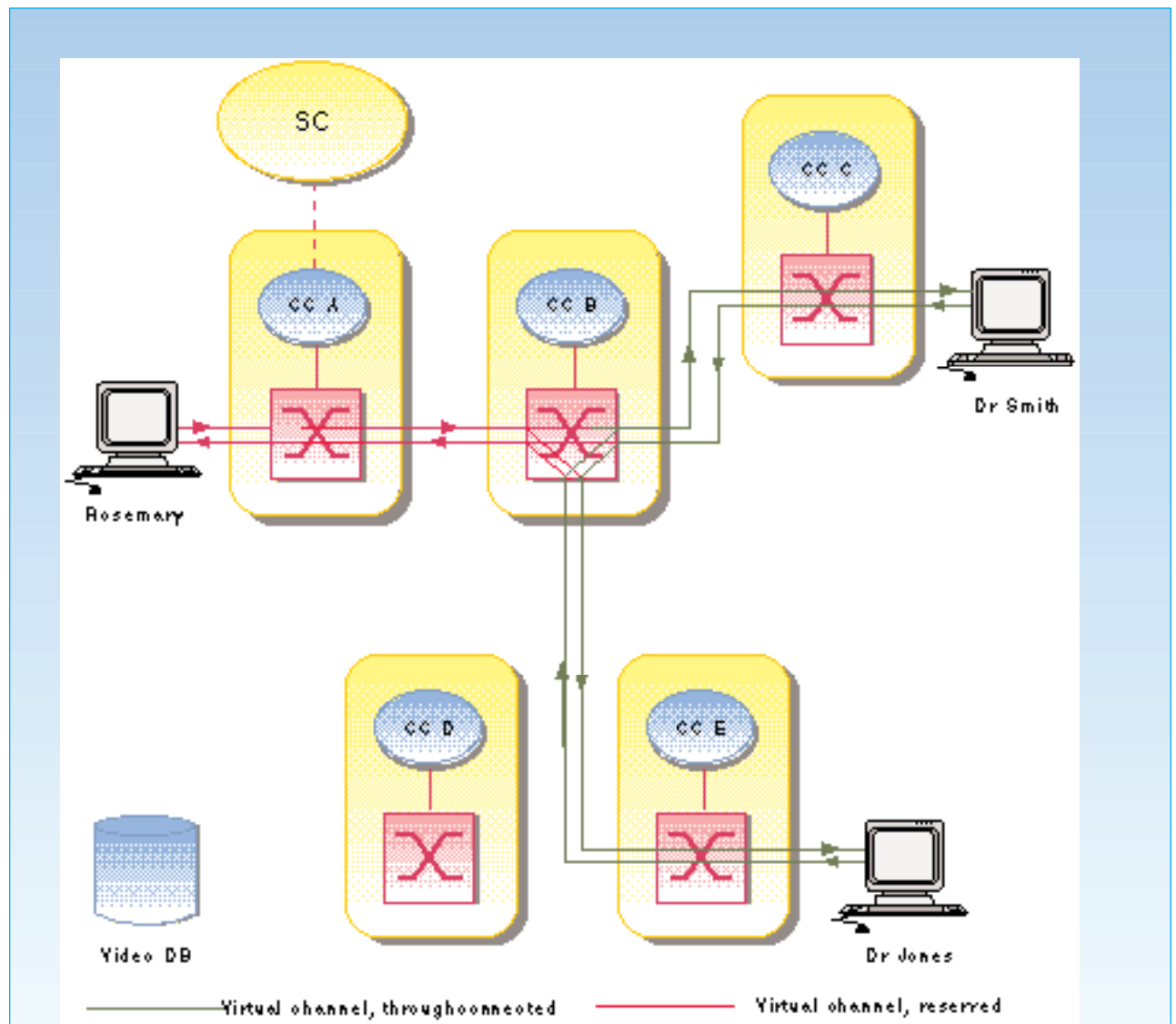


Figure 15 Three-party configuration

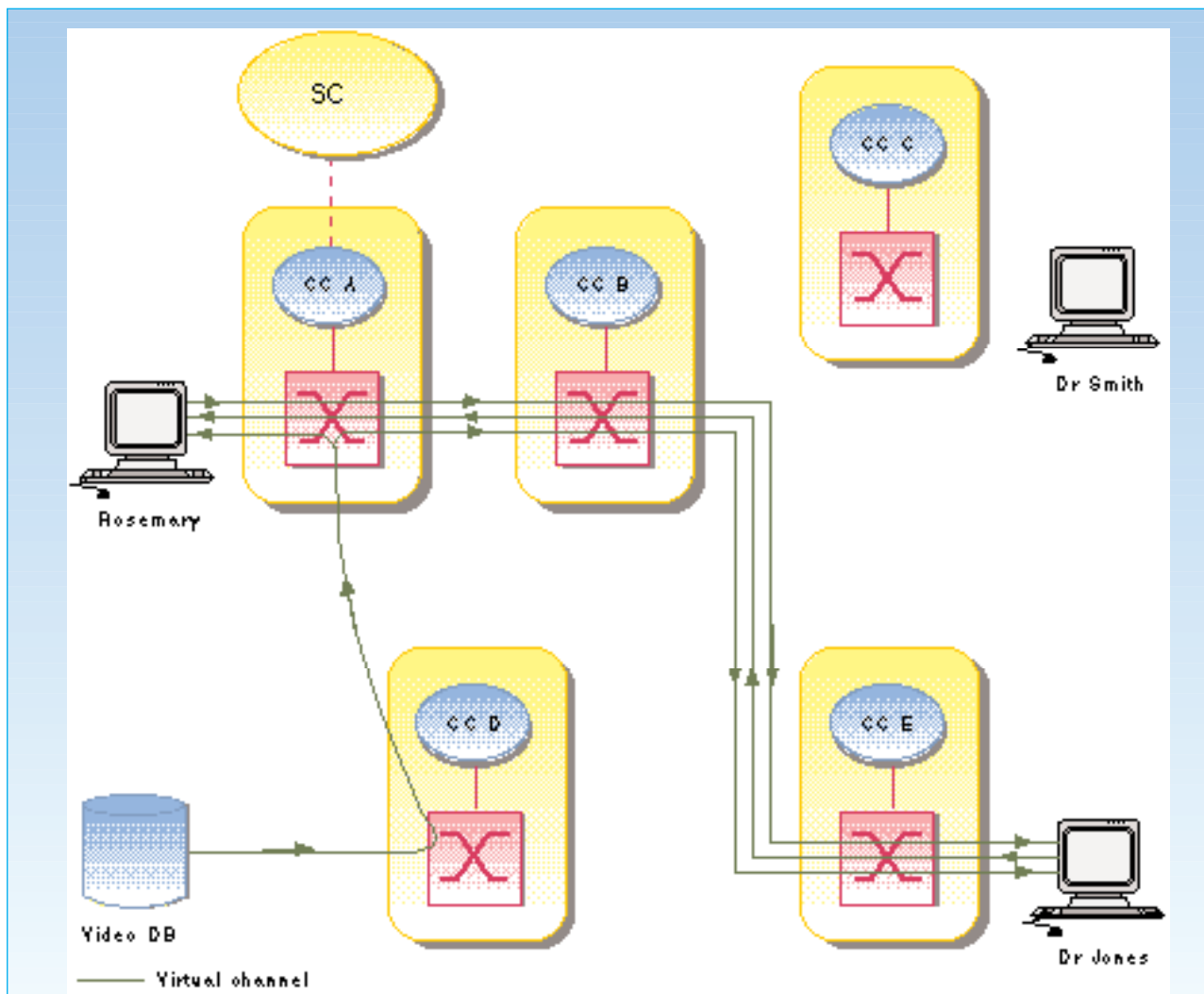


Figure 16 Three-party configuration including video distribution

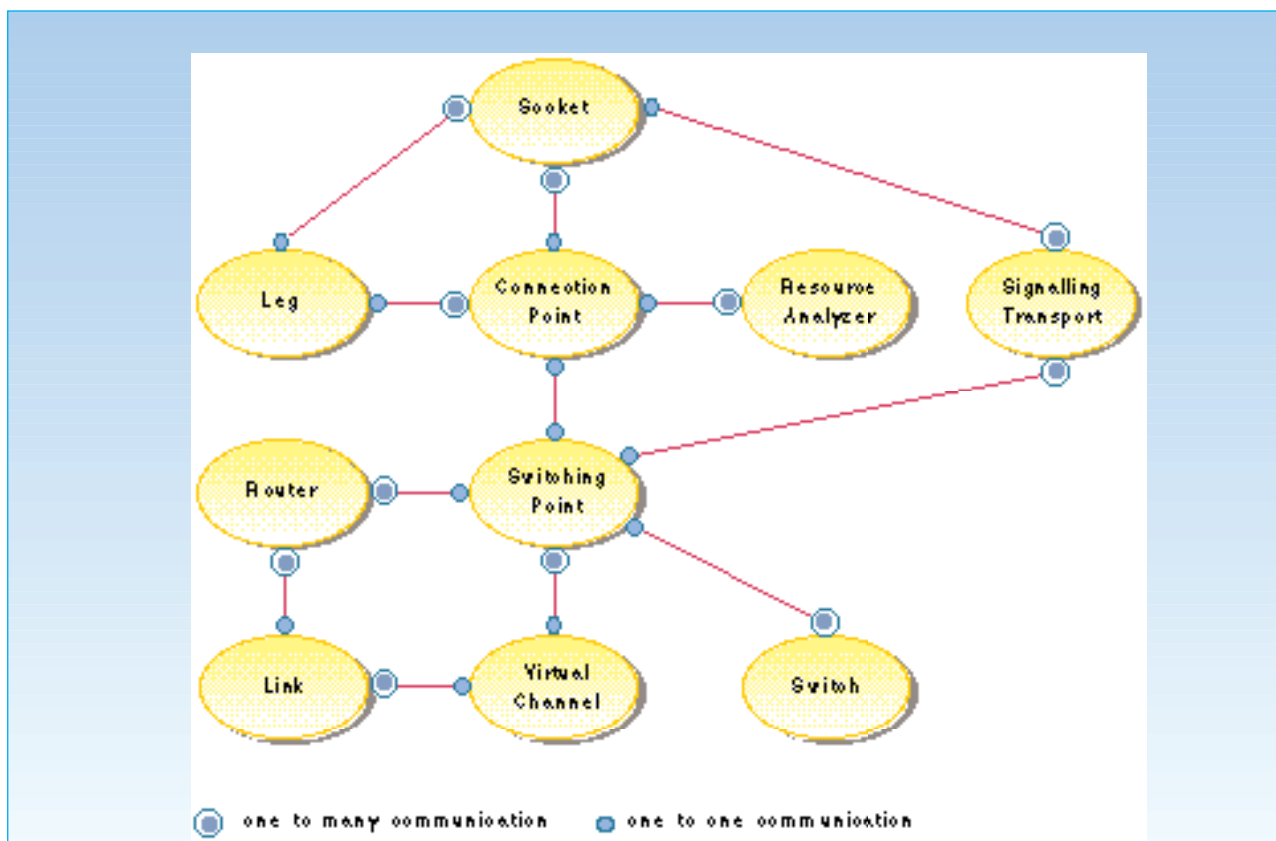


Figure 17 Software structure in one CC node

The *create\_cp* message defines a connection point reference.

When the first *create\_leg* message is received, CC-A starts allocating network resources corresponding to this leg. Since this is a bidirectional leg, two virtual channel connections from exchange A to doctor Smith's terminal are required, one in each direction.

Each of the CC nodes participating in the connection establishment performs the same procedure, in our implementation consisting of three main steps:

- Routing
- Link resource reservation and VCI negotiation
- Switch resource reservation.

Algorithms for routing and resource allocation in an ATM network is currently being developed in international organisations and research laboratories, including NTR (2, 3).

#### *Routing:*

In our current implementation we have chosen a very simple routing method. Routing implies identifying the next physical link to be used for the connection, the identified link leading directly to the destination terminal or to another exchange.

The two needed virtual channel connections are created in parallel with identical routes.

The set of possible links is found by looking up in a static routing table. Only links with available bandwidth are considered. The links are ranked according to number of hops to the destination address. This number is found in the routing table. A link is selected according to the following rules:

- 1 The link with the lowest number of hops is preferred.
- 2 If several links have equal number of hops, one of them is arbitrarily chosen.

If, as in this case, two virtual channel connections are needed, they are always created in parallel, with identical routes. This constraint only complicates the described routing algorithm slightly. The motivation for this decision is explained below.

#### *Link resource reservation and VCI negotiation:*

Once the physical links have been chosen, the internal data structures of the CC node is updated, to reflect that the required bandwidth on the links have been reserved.

VCI negotiation implies that the CC node exchanges messages with the neighbouring CC node or terminal to reach an agreement on the VCI values to be used on the selected physical links.

#### *Switch resource reservation:*

As explained in section 2.1, the main function of the switch is to interconnect incoming virtual channel links with outgoing virtual channel links. When the VCI negotiation is completed and the virtual channel links are successfully established, the CC node completes its task by setting up the connections through the switch. The connections are only reserved through the switch, not through-connected. User information is not allowed to flow on the connections until the appropriate *join* messages are issued by SC.

Note that CC-A is not able to set up connections through its switch when the first leg is created, since the switch at this point in time is an end-point for the connections.

Eventually, the connections will reach doctor Smith's terminal, completing the actions resulting from the first *create\_leg* message.

The second *create\_leg* message from SC makes CC set up virtual channel connections to Rosemary's terminal.

To summarise, the result of the first four messages is reservation of all network resources necessary for the communication to take place between doctor Smith and Rosemary (i.e. a point-to-point bidirectional connection).

#### *Through-connect:*

The effect of the *join* messages is through-connect of the connections in the switches. This will open for the flow of user information on the connections.

When the first *join* message is received, CC-A only stores the *join* request. Since, in general, more than two legs may be allocated to the connection point, CC does not have enough information to per-

form through-connect until it has two end-points, i.e. two joined legs. Thus, when the second *join* message is received, the virtual channel connections will be through-connected in all switches. The CC nodes will act on response of a *connect* message, which is created by CC-A, and travels through all CC nodes along the route.

The resulting configuration is illustrated in Figure 14. Voice communication can now take place. The video connections are established in a similar way (not shown).

### 3.3.2 Three-party configuration

To illustrate how CC supports configurations with more than two legs allocated to a connection point, we turn to what happens as doctor Smith temporarily puts the connection to Rosemary on hold in order to consult doctor Jones.

This example shows how the constraint that only two bidirectional legs may be joined to a connection point is used by CC to reduce the number of virtual channel links allocated. The idea is that several legs may share the same virtual channel links. The point-to-multipoint and multipoint-to-point capabilities of the switch, as well as the capability of differentiating between reserved and through-connected connections are also necessary ingredients in this solution.

The following sequence of messages are sent from SC to CC (again, only showing the speech part):

*split (leg2)*

*create\_leg (cp1, leg3, speech, bidirectional, Dr Jones)*

*join (leg3)*

The *split* message triggers the reverse operation of the previous *join*, i.e. the virtual channel connections are disconnected (i.e. returned to the *reserved* state) in all the switches. User information (speech) can no longer flow between Rosemary and doctor Smith. All network resources remain reserved, however.

The procedure for connection establishment was described above. When the *create\_leg* message for *leg3* is received, this procedure is performed again. This



**Socket:** This object represents an interface between a connection control node and the SC domain. Socket coordinates operations from SC and forwards them to objects capable of performing the requested services. This object resides within a CC node. Service Control will initialise a Socket instance for every session with the CC node. The messages sent to Socket from Service Control are identical to the service primitives listed in section 2.3.2 (“The SC-CC Interface”).

**Leg:** This object represents a communication path from a connection point towards an addressable entity in the network (e.g. an end-user terminal). A Leg is always associated with a Connection Point. Leg is responsible of storing the information characterising a leg. Socket may create and control several Leg instances.

The most important messages forwarded from Socket to Leg are:

- *create*
- *join*
- *split*
- *modify*
- *free*

**Connection Point:** This object plays several roles. As seen from Socket and Leg, Connection Point represents an interconnection or logical bridge allowing information to flow between legs. Furthermore, this object is responsible for interpreting (in terms of ATM parameters) the virtual switch picture and manage the ATM connections through the switching node. As seen from Figure 17 the Socket may control several Connection Point instances and several Leg instances will be associated with one Connection Point object.

The most important operations forwarded from Socket to Connection Point are:

- *create\_cp*
- *free\_cp*

Notifications sent from Leg to Connection Point are:

- *create\_leg*
- *join\_leg*
- *split\_leg*
- *modify\_leg*
- *free\_leg*

**Resource Analyzer:** On requests from Connection Point, Resource Analyzer performs a mapping of user oriented (technology independent) description of network resources into ATM specific parameters. One CC node will contain one instance of Resource Analyser.

The following operation is defined on Resource Analyser:

- *get\_bitr*

**Switching Point:** This object represents a switching point of a single ATM connection. The connection may be unidirectional or bidirectional. A connection through several switches will have one Switching Point in each CC node. Switching Point objects are cooperating to set up virtual channels between the switches. One Switching Point may create and control several Switching Points in neighbouring CC nodes.

The following messages are sent from Connection Point to Switching Point:

- *create\_sp*
- *setup\_leg*
- *connect*
- *disconnect*
- *release\_leg*

Messages sent between Switching Points in different CC nodes are:

- *create\_sp*
- *setup\_vc*
- *release\_vc*
- *setup\_leg*
- *connect\_leg*
- *disconnect\_leg*
- *modify\_leg\_bitr*
- *release\_leg*

**Router:** Router represents a manager of all external links associated with a switch, and performs the routing of connections not to be terminated in this switch. Based on destination address and bandwidth/Quality of Service requirements, Router is capable of identifying the next switching node and selecting a link to this node. One CC node will contain only one Router instance.

The most important message sent from Switching Point to Router is:

- *select\_route (dest\_addr, conn\_type, peak\_bitr)*

**Link:** This object represents a link connected to the switch. Link is responsible of administrating the resources (bandwidth and VPI/VCI values) associated with the link.

Messages sent from Virtual Channel to Link are:

- *reserve\_bitr*
- *modify\_bitr*
- *assign\_vci*
- *release\_vci*
- *release\_bitr*

**Virtual Channel:** This object represents a single virtual channel on one of the links going into or out of the switching node. Virtual Channel is responsible of storing information characterising the virtual channel. Several Virtual Channel instances will be associated with a Switching Point instance. The most important messages sent from Switching Point to Virtual Channel are:

- *create\_vc*
- *set\_vci*
- *modify\_bitr*
- *release*

**Switch:** This object represents an interface between a CC node and an ATM switch. Switch Agent provides CC with a set of hardware independent switching services to control the interconnection of virtual channels. One Switch Agent instance will be associated with one CC node.

Messages needed for reservation, modification and through-connect of switching resources for an ATM connection are:

- *reserve*
- *connect*
- *modify*
- *disconnect*
- *release*

**Signalling Transport:** This object represents a generic sender and receiver of asynchronous/external messages. Sending messages to external collaborators implies packing the messages into network datagram packets and interacting with the communication system to send the packets. Receiving messages from remote collaborators implies fetching the packets from the communication system, identifying the receiving objects, unpacking the messages and sending them to the identified objects.

Messages defined are:

- *send*
- *receive*

Figure 18 Object descriptions

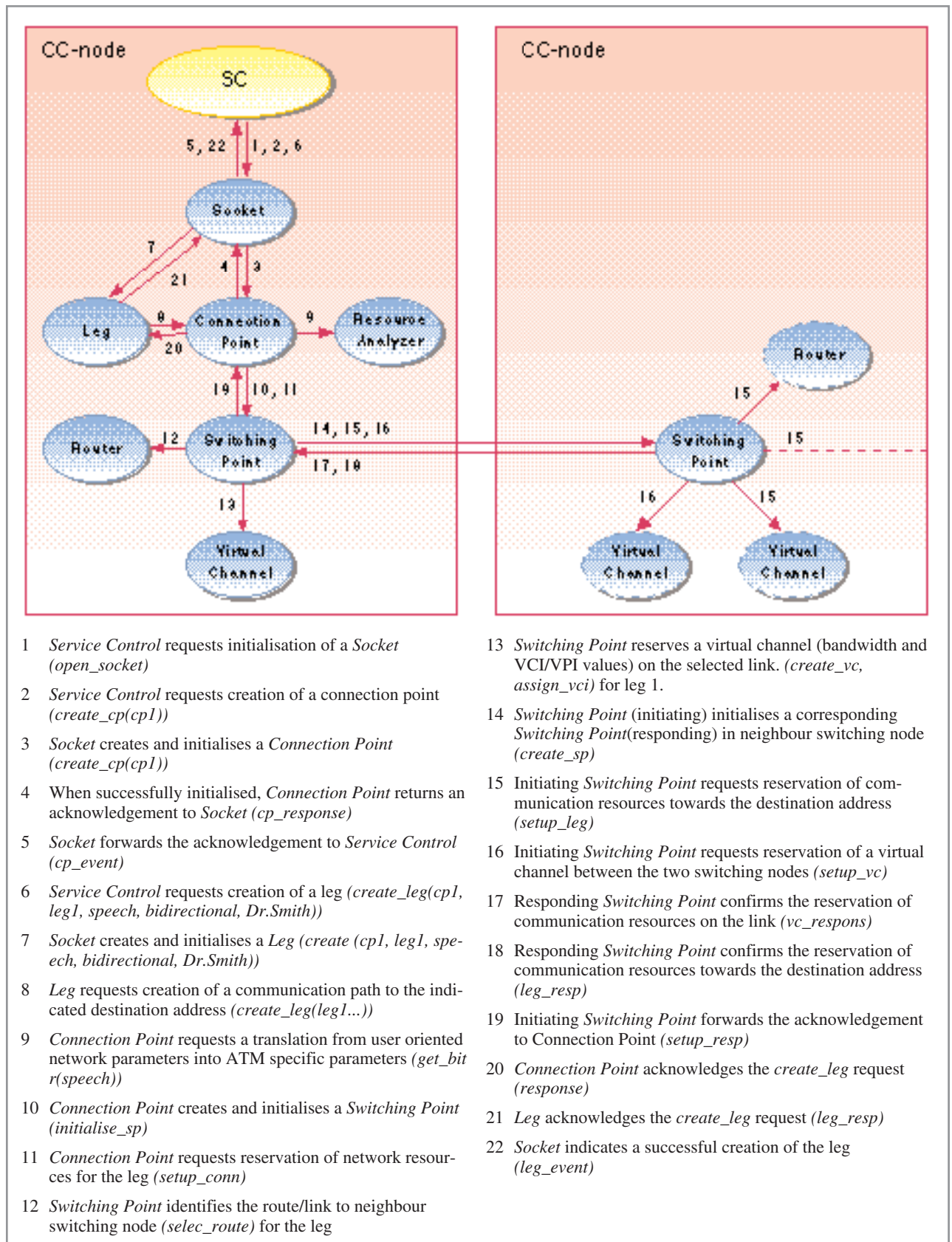


Figure 19 A message flow scenario (creation of a connection point and the bidirectional audio connection towards doctor Smith)

time, however, some additional points can be made. The result of the routing step in CC-A, makes it clear that B is the most suitable next node, i.e. a virtual channel link from A to B is needed. There is, however, already allocated a virtual channel link from A to B, for *leg1*. As pointed out above, there is no reason to allocate a second one. If a second link was allocated, the constraint of maximum two joined legs (i.e. point-to-point communication) implies that at any point in time, at most one of them would be used.

The resulting configuration is shown in Figure 15. Only one link is allocated between switches A and B. Note that through switch B, each of the three incoming virtual channel links are connected with two outgoing links. Thus, both point-to-multipoint, and multipoint-to-point relations are found. Note that this rather complex situation is only reserved, not through-connected. The point is that sufficient network resources have now been reserved for any pair of partners to communicate.

As mentioned above, the two virtual channel connections allocated to realise a bidirectional leg are always routed the same way (in parallel). The reason is seen from the example. If the two connections between Rosemary and doctor Smith had been routed separate ways when the first two legs were created, the idea of different legs sharing links would be considerably more difficult to realise.

The last join message implies that user information should be enabled between the doctors Smith and Jones. The connections are thus through-connected in the switches C and E, as well as the appropriate parts of the complex connections in switch B, as shown in Figure 15.

Finally, to illustrate how the video distribution configuration is realised in terms of ATM virtual channel connections; we move to the point in the call where doctor Jones decides to use the video database. SC requests the bidirectional video connection point and associated legs to be deleted (using *free\_leg* and *free\_cp* messages), and requests a unidirectional connection point with associated legs to be created. The message sequence is as follows:

```
create_cp (cp3)
create_leg (cp3, leg7, video, upstream,
           VideoDB)
create_leg (cp3, leg8, video,
           downstream, Rosemary)
create_leg (cp3, leg9, video,
           downstream, Dr Jones)
join (leg7)
join (leg8)
join (leg9)
```

The actions performed by CC as the messages are received, follow the scheme explained above, and will not be repeated here. The resulting configuration is shown in Figure 16.

## 4 Internal structure of a CC node

In the trial network, one CC node is controlling one ATM switch. The CC node interfaces other CC nodes, SC nodes and control software in CPNs and end-user terminals. The signalling network interconnecting different control nodes is implemented by standard Ethernet, while the CC nodes are implemented in Sun workstations. Figure 5 shows the physical configuration of the control system.

This section gives a more detailed description of the software in one CC node. A flexible and simple structure, picturing the underlying switching system, is obtained by applying object oriented principles. In our implementation, OOR-ASS (5), an object oriented modelling method is combined with the programming language Eiffel. (4)

Figure 17 shows the basic object types in a CC node and their interconnection, Figure 18 contains supplementary object descriptions and Figure 19 shows a message flow scenario. The message flow scenario shows the object interaction needed to reserve the speech leg towards doctor Smith (*leg1* in our service example). To simplify the message flow scenario, objects handling communication across external interfaces are not included. In the figures the term message is used in the object-oriented sense, i.e. to "send a message to an object" means the same as "request a service" or "invoke an operation" on it.

As seen from Figures 17 and 18, transport problems are encapsulated in the *Signalling Transport* and *Switch* objects. To avoid looking at B-ISDN signalling protocols in our first prototype, the Ethernet is used for signalling transport. An integration of signalling traffic in the ATM network requires updated functionality in the *Signalling Transport* and *Switch* objects, but has no impact on the overall object model. Thus, our CC model is independent of the signalling network chosen.

## 5 Conclusion

The B-ISDN connection control system at Norwegian Telecom Research offers powerful and flexible handling of various connection configurations needed to support B-ISDN services. The use of IN concepts and object oriented principles have made this possible.

By adopting the virtual switch concept in the Connection Control Socket Model developed by ETSI/NA6, our control system fulfills some of the main IN objectives:

- The control system allows a network independent execution of services.
- The control system provides the services (call instances) with a simple consistent view of the underlying network.
- The control system is powerful in terms of providing a large variety of connection configurations. However, due to limited signalling multiplexing capabilities in the ATM switches, some restrictions are put on the possible combinations of legs and connection points.

As described, the CC domain performs the mapping from legs and connection points to ATM connections and physical switching points. An efficient use of network resources seems to be obtained by

- allowing legs associated with the same connection point to share communication resources (virtual channel links) in the ATM network
- mapping a connection point to a distributed set of switching points.



To focus on the functionality of the connection control system and avoid looking into B-ISDN signalling protocols, standard Ethernet has been used as a dedicated signalling network. However, our CC model decouples completely signalling and transport aspects, and an integration of the signalling traffic in the ATM network will have no impact on the software structure in a CC node.

## References

- 1 ETSI DTR/NA-6001. Draft Technical Report, *Intelligent Network: Framework, Version 2*. September 1990.
- 2 Hemmer, H et al. *Trafikkfunksjoner i ATM-nett*. Kjeller, Norwegian Telecom Research, 1992. (TF N1/92.)
- 3 Pettersen, H. *Dirigering og ressursallokering i ATM-nett*. Kjeller, Norwegian Telecom Research, 1990. (TF F23/90.)
- 4 Meyer, B. *Object-oriented software construction*. Englewood Cliffs, N-J., Prentice-Hall, 1988. ISBN 0-13-629031-0.
- 5 Reenskaug, T et al. *OORASS: Seamless Support for the Creation and Maintenance of Object Oriented Systems*. Oslo, Taskon, 1991.
- 6 Handel, R et al. *Integrated broadband networks: An introduction to ATM-based networks*. Reading, Mass., Addison-Wesley, 1991. ISBN 0-201-54444-X.
- 7 *CCITT Recommendation I.150*. B-ISDN ATM Functional Characteristics. Geneva, 1991.
- 8 *CCITT Recommendation I.211*. B-ISDN Service Aspects. Geneva, 1991.
- 9 *CCITT Recommendation I.311*. B-ISDN General Network Aspects. Geneva, 1991.

# Some issues of security and privacy in intelligent networks

BY KLAUS GAARDER

681.324.004.4  
621.39.05

## Abstract

Security and privacy in intelligent networks is treated as a case of security and privacy in distributed systems. We consider the concepts of security policies and security domains. The enforcing of security policies in intelligent networks is given special attention, and interdomain problems are seen as one of the major challenges. Authentication, authorisation, data security and security management are discussed in broad terms and with some examples from services like UPT and VPN. We finally consider formal methods of description and specification combined with object orientation as possible tools. The paper aims to give an initial analysis of some security issues in intelligent networks.

## 1 Introduction

Intelligent networks will offer advanced information processing and networking of computers directly to the *general public*. The terminal equipment in our homes will probably consist of “work-station-like” multi-purpose machines. As increasingly advanced services pervade our daily work we will need to have assurance of their *security properties*. That is, we must feel assured that information we want to keep private *stays private*, and that information we trust and depend on to be correct, *stays correct*, etc. Put in terms of what is to be discussed in this paper the intelligent network must be able to *enforce a very wide variety of security policies*, ranging from rather simple to quite restrictive.

The intelligent network and its services must provide a sufficient variety of security functionality to satisfy *every user*. This is so because the intelligent network is not just a part of the future telecommunications network but *it is the future telecommunications network*.

As indicated above, the security problem is essentially the problem of *enforcing a security policy*. Many texts dealing with security aspects of computer systems tend to give the impression that security is mainly about encryption and secrecy (confidentiality), security levels, authentication protocols, etc., or that absolute standards and references exist, which is not true. Security is a truly relativistic subject, and the frame of reference is always given by, and only by a security policy. Note that “national security” provides no absolute reference, it is just another (time dependent) policy, although for many cases it is used as some sort of “default” policy or baseline.

Throughout the paper we assume some familiarity with the intelligent network concepts and refrain from any elaboration on these unless necessary for the clarity of exposition.

## 2 Intelligent network = distributed system

One of the basic ideas of the intelligent network is the separation of *service logic* from *switching logic*. An effective abstraction of this is to view the intelligent network as consisting of two logically separate “domains”, the *service domain* (*S*) and the *switching domain* (*X*). Services reside in *S* while connections and other communications specific facilities are placed in *X*. A well-defined interface between *S* and *X* takes care of the interactions between the two domains.

A close analogy is an application on a workstation with network communications functions. The application is running on your computer, supported by the operating system and communications software and hardware. It may, or it may not communicate with other applications without the user being explicitly aware of this. Most important is that you do not worry about how this happens, as long as it works satisfactorily.

This view reduces *the switches and network* to a basic communications “engine” acting as a provider of *connectivity*. The services need only know that this connectivity is there when they need it and they can ask for communication

whenever they want to. Still the communications can be seen as an “add-on” to most services (though some would look silly without, e.g. Plain Old Telephone Service, which basically is only communication).

The services can in general be viewed as *distributed applications* on this network (5). (A service which is centralised can be viewed as trivially distributed on one node.) In general, there will be different situations where distribution is natural or preferable for various reasons. However, we may see that many services of the intelligent network will be of limited interest if they are *not* distributed. The Universal Personal Telecommunications (UPT), Universal Access Number (UAN) and Virtual Private Network (VPN) services are prime examples of this. This leads us to conclude that in order to have an unrestrained view the intelligent network is best considered as an *open distributed system on the public switched network*.

An immediate conclusion is that security and privacy in intelligent networks essentially reduce to security and privacy in distributed systems. We cannot see technically new security problems in the intelligent network that are not present in a general open distributed system. The

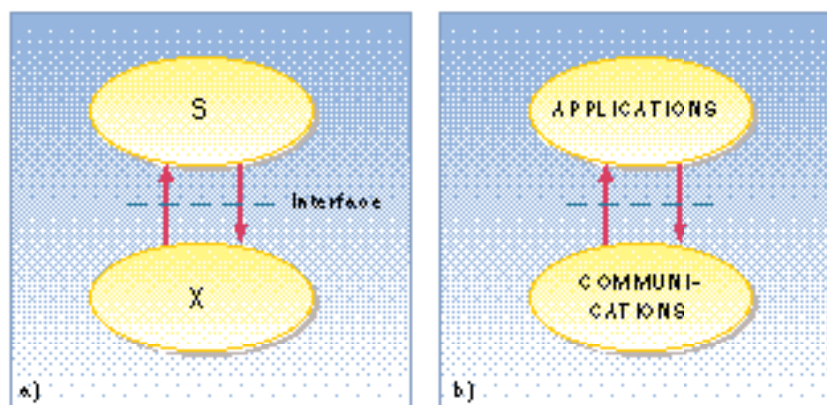


Figure 1 a) A graphical rendering of the logical separation into Service domain and Switching domain  
b) Another view of the separation

general security issues are the same even if the applications are to some extent different. The differences between a completely general open distributed system and the intelligent network lies in the policy structures we may see. This will be further elaborated on below. We are then in a position to discuss security and privacy in intelligent networks on a firm basis, since security in distributed systems is a well established area of research. Henceforth we will consider results on security in open distributed systems as equally valid for intelligent networks security. Whenever “security”, “security policy”, etc. is used, it is to be understood in this context unless otherwise indicated.

### 3 Security policies

As indicated in the introduction the term “secure” is a relativistic term. Questions like “Is this secure?” or “Has there been a security breach?” are meaningless or at best very hard to answer unless we are given a frame of reference, a security policy. The following definition of security policy is from (1):

*Definition 1 (Security Policy):*  
A set of rules which define and constrain the types of security-relevant activities of entities.

The definition captures the fact that a security policy is a collection of statements on

- what is considered security relevant, and
- how to protect these things from damage or compromise.

Note that in order to avoid a circular definition the term “security” in this definition has to be interpreted as a qualitative concept established by those who are defining the security policy. What this means is always hard to decide, and is one of the crucial questions we have to answer when designing a security policy. In regular business contexts the usual practice will often be to put some economic measure on what you are doing and decide what it will cost in terms of labour and money to restore what has been compromised or lost. In some cases this is easy and in other cases it is almost impossible.

The only way we can “define” security quantitatively is *by reference* to some given security policy:

*Definition 2:*  
An object is secure if and only if its state is in compliance with the security policy enforced.

Security policy is violated if an object fails to meet one or more of the requirements set by the security policy.

This definition is not very precise (the state of an object is unclear), but it carries the necessary message that *security is always relative to some security policy and nothing else*. There does not exist a thing such as *absolute security*, simply because there is no reference.

A nice analogy is the law of ordinary society. What is allowed in one state may be prohibited in another and thus “breaking the law” is a relative concept. This also means that if no security policy exists, no violations of security can exist (no law means no lawbreakers)!

#### 3.1 Security domains

The concept of a *security domain* has turned out to be very useful in structuring complex security scenarios. A *security administration* as defined by (1) is

*Definition 3:*  
A human authority which establishes a security policy and identifies the entities to which the policy applies.

The security domain is then associated with security administrations in the following definition of a security domain (1):

*Definition 4:*  
A security domain is a set of entities (objects) that is subject to a single security policy and a single security administration.

Security domains are the domains of jurisdiction of a security administration. The way they are defined, security domains may form hierarchies. They will either be disjoint or one completely included in the other (since it seems reasonable that no object may be partially under purview of one policy and partially another, since nothing guarantees that these two policies are consistent with one another). The world, seen from a security point of view, may now be partitioned into a patchwork of security domains,

each defined by its security policy. To decide which security requirements apply to an object, decide which domain it belongs to and check the security policy for that domain.

#### 3.2 Interdomain problems

Problems occur when objects move from one security domain to another. If the two domains are part of the same hierarchy there is usually some pre-defined procedure to take care of this. If the two domains are disjoint then we may have what we call an *interdomain problem*. The two policies may be incompatible in some respects or even incomparable in others. Traditionally such problems are solved by negotiation in each case or each class of cases. However, in our days of increasingly fast electronic business this is not always possible or desirable, so the problem will have to be solved by other means.

The reason why interdomain problems are so hard to deal with is due to several factors among which the lack of possible standardisation is one. We believe it is impossible, maybe even meaningless to standardise security policies. The only possible subject of standardisation is on the implementation side. Some of the mechanisms for implementing and enforcing security policies may be given standard forms. The ECMA Privilege Attribute Certificate (PAC) and Control Attribute Package (CAP) are attempts at this for open distributed systems, hence also applicable to intelligent networks. We refer the reader to (2) and (1) for details.

We believe interdomain problems will be one of the most challenging security issues in intelligent networks in particular and distributed systems in general. This is due to the fact that interdomain activities is not an exception but a rule in doing business.

### 4 Security in the intelligent network

In this section we consider security in the explicit context of the intelligent network. First we discuss general security policy issues. Then we move on to discuss authentication, authorisation, data security and security management in broad terms, with some examples from e.g. UPT and VPN.



## 4.1 Policies in the intelligent network

For the remaining part of this paper it will be assumed that for any situation, we are under the jurisdiction of some security policy, and every object will be assumed to belong to some security domain. This is just to ensure that talking about security will always be meaningful.

There will be security domains for the different actors on the scene, like service provider, subscriber, user and network operators, etc. Certain relations between these domains (or policies) will be given by some unavoidable physical facts of the network. The network operators will typically require their policies to be enforced with regard to the security of the network operations, we could call this the “network policy”. Hence, as all services use this network, they will have to comply with this network policy in addition to whatever policy they have of their own. Network policy may be seen as a bottom (or top) level of a policy hierarchy in the sense that all services must comply with it, their own policies considered a “refinement” of it.

The exact relations between the different policies will be important for the way we handle security in intelligent networks. Among other things this will decide the extent of interdomain problems we will encounter. Consider the following policy structure, where  $P_i \subseteq P_j$  means the first policy is included in the second, i.e. all rules of  $P_i$  are in  $P_j$ .

$$P_0 \subseteq \left\{ \begin{array}{l} P_1 \subseteq \left\{ \begin{array}{l} P_{1,1} \\ P_{1,2} \end{array} \right. \\ P_2 \\ P_3 \subseteq \left\{ \begin{array}{l} P_{3,1} \\ P_{3,2} \end{array} \right. \end{array} \right.$$

Here we may have well defined relations between  $P_0$  and all other policies, since they are all including  $P_0$ . The more refined (with respect to  $P_0$ ) policies  $P_{1,2}$  and  $P_{3,2}$  may have to go to  $P_0$  to find a common ground to resolve interdomain activities. A possible solution in this case could be to define a separate *interdomain policy*, dictating rules for interaction with other domains. (A simple such policy is “If  $P_x$  is a sub-policy of this policy it’s

OK, else NO INTERACTION”). In the absence of such simple general rules, an interdomain policy could be quite involved and maintenance will be a major problem.

We believe that the policy structures of the intelligent network will be significantly more complex than the example above. It should be a prime task to investigate how we can best accommodate these various structures.

## 4.2 Authentication

The authentication problem is the problem of verifying the identity of entities. Or as (1) defines it

*Definition 5 (Authentication):*  
The process by which the identity of an entity is established.

Authentication can be done in a multitude of ways, some of which are simple (e.g. simple passwords, PIN codes) and some which are rather complicated (e.g. zero-knowledge protocols or interactive proof systems). In recent research on security, authentication has been receiving growing attention. This is due to the increasing importance of open and distributed systems. In such systems you cannot know at all times who will try to access the system. The other extreme is a system which is only accessible from a single physical location with strongly restricted access. In the intelligent network we must ensure that we can authenticate several kinds of entities, ranging from human users via physical terminal equipment to processes. The requirements for authentication in each case will be stated in an authentication policy to be enforced. The main problems for services providers will be to accommodate all the different varieties of authentication procedures which may be required at all times and all locations. Some standards have been proposed, like the X.509 Authentication Framework protocols. However, formal analysis of these protocols have revealed weakness (e.g. (9, 6)).

Nevertheless, the X.509 recommendations probably give the best starting point for a standard distributed directory service containing public key information and certificates for authentication.

## Authentication in UPT

The UPT service will present a typical authentication problem. In a fully developed UPT service it is vital to know who exactly is using the service. A primary reason is billing. It should be exceedingly difficult for a user Alice to masquerade as another user Bob, using his UPT service and making him pay the bill.

The authentication issues in UPT have been recognised by ETSI. It is reasonable to assume that the experiences with the authentication issues in the GSM system will have an important effect on the work with authentication in UPT.

As a UPT user moves around she will pass through differing network segments and thus possibly differing security policies will be in effect in these segments. She may also choose to act either as a professional or as a private person, thus filling different roles with different security policies, possibly requiring different authentication schemes.

### UAN

One particular problem which may be associated with UAN in particular is the authentication of the *service*. That is, if Alice calls a UAN she might be interested in being assured that this is in fact the real UAN she wanted, not some fake. Depending on the underlying business of the UAN subscriber this can become a real issue.

### VPN

There is no reason to regard a virtual network as much different from “real” networks with their requirements for access control. Security may be split between the security in the actual physical network and the VPN service. The enforcement of a VPN security policy may present a significant challenge. VPN is typically a service which in our opinion will be impossible to bring to market without advanced security features present from day 1.

### Who authenticates who?

As authentication is a two-party process, there is always possibly a need for authentication for every combination of pairs. If we assume that in the intelligent network we have  $n$  generic entities we can at worst risk  $n^2$  possible authenti-

cation situations requiring differing treatments. However, we believe the following ones to be among the most probable and common (User is a service user agent or a human user in the sense of end user):

*User - Service* (a user authenticating to a service)

*Service - User* (a service authenticating to a user)

*User - Network* (a user authenticating to a network, real or virtual)

*Network - User* (a network authenticating towards a user)

*Subscriber - Provider*

*Provider - Subscriber*

Note the asymmetries arising from possible differences in policy, i.e. the fact that “User - Service” may be different from “Service - User” authentication. Differing policies and requirements will dictate how each of these authentications are to be performed. Another issue which will have to be resolved is the definition of these entities, i.e. what is a “user”, “service provider”, “service subscriber”, etc.

### 4.3 Authorisation

Authorisation is often confused with authentication, because in many situations which we normally encounter there is apparently no difference. Let us quote the following definition from ECMA Standard 138 (1):

*Definition 6 (Authorisation):*  
*The process by which an access control decision is made and enforced.*

If we follow this definition then *authorisation* is quite different from authentication, and also from ordinary use of the word. Authorisation is the very *process of granting or denying access* to resources. That is, an entity is not “authorised” until it is actually granted access. Authorisation is of course tightly coupled to authentication. Some security policies grant access based solely on authentication, while others require further requirements to be fulfilled (like being in possession of certain credentials).

Authorisation policy and authentication policy together form the core of what is called an access control policy. The access control policy decides who can access resources and how (e.g. read, write, execute, etc.). In the intelligent

network we will need access control policies for all resources which should have some limitations on their accessibility. Exactly how these are implemented may vary, but we suggest using the privilege attribute and control attributes concepts from the ECMA STD 138 (1). These are general constructs designed for use in open distributed systems, and hence should be suitable also for intelligent networks.

### Particular services

It is hard to point to any particular service on this issue. Any service which may be used to access resources in ways that can be considered harmful will need access control of some sort, which will be decided by the security administration of the domain to which the resources belong. Subscriber and service profiles will be typical entities in need of access control, as will the basic network entities.

Authorisation is often based on *ownership to information*. Thus, to decide who owns any particular piece of information can become an important issue. Ownership is not always obvious. If we create documents in our business these will typically be regarded as owned by whichever company we work for, or even a customer of this company. Internally in the company ownership may be more refined, relating to smaller units all the way down to persons. Internally the person producing the document may be considered the owner.

The important issue in the intelligent network is to be able to accommodate any such variety of authorisation policy, not to decide which should exist and which should not!

### 4.4 Data security

Data security in general involves two aspects: *confidentiality* and *integrity*.

#### Confidentiality

To keep certain things from being known to unauthorised persons is a fact of daily life with smart cards, credit cards, etc., passwords and PIN codes of all sorts. (2) defines confidentiality as

*Definition 7 (Confidentiality):*  
*A security property of an object that prevents its existence being known and/or its contents being known. This property is*

*relative to some subject population and to some degree of security.*

Confidentiality is the classical security problem and thousands of work-years have been spent to find methods of protecting information from unauthorised disclosure. This is the encryption or cryptography business. Equally hard efforts have been put in to break through other people’s protection, this is cryptanalysis.

In the intelligent network, confidentiality will be a question of *communications security*. We regard the confidentiality problems of the customers internally as outside our domain of discourse. Intelligent network services involving transfer and manipulation of information considered security sensitive by some customer will have to provide some sort of protection compliant with the security policies in effect. Usually this will imply the use of encryption under keys known only to authorised entities. Here we run into the more delicate sides of the security business. Due to the importance of cryptographic techniques in international intelligence and military business, heavy restrictions exist in many states. In some states the transfer of encrypted information in the public network is prohibited. These issues are *not* easily resolved and may put severe limitations on some uses of encryption in intelligent network services.

#### Integrity

A definition of *integrity* to be found in (2) is

*Definition 8 (Integrity):*  
*A security property of an object that prevents or is used to prevent its condition of existence being changed and/or its contents being changed. This property is relative to some subject population and to some degree of security.*

Integrity of information or other items is less obvious to us. Often we have unconscious mechanisms for assessing the integrity of things, like traces of tampering, obvious flaws which should not be there, etc. However, if Alice sends Bob an electronic message, or calls Bob on the phone, how can he be sure that her message has not been modified on its way or that it is actually Alice talking? These are integrity questions. The last issue could be solved by authenticating

Alice to Bob. The first could be solved by means of e.g. a *digital signature*, which is a cryptographic means of ensuring the integrity of information.

We may find that the demands for maintaining high integrity of information in the intelligent network are larger than those for confidentiality. It is often much more important that the information you receive is known to be uncorrupted than that it should be kept secret. Of course, encryption for secrecy implies uncorrupted information if the key has not been compromised. However, bulk encryption is time consuming and expensive, while integrity can be achieved by simpler means. Thus bulk encryption will often be “over-kill”.

#### 4.5 Security management

Security management concerns the actual operational aspects of a security administration. It is one of the most critical functions in any security system needing to be absolutely trusted by all entities under purview of the particular policy (at least). We must take care not to confuse the ones responsible for security management with the policy makers. The relation is much like the one between legislators and the police. Security management in the intelligent network will be an awesome task.

### 5 Specifying security in IN

A primary reason for problems with implementing security in computer systems is that security questions often are considered much too late in the design and engineering process. Introducing security at a late stage is invariably a difficult, time consuming and expensive task. In the intelligent network we will have services which are depending on necessary security services in order to be interesting to the market. No business customer in her right mind would subscribe to a VPN service without the right security being available. This means that service providers must be able to supply security from the start. This means further that security must be an integrated part of the specification of a service. No service can be seen separated from its security aspects.

#### 5.1 Security services

Based on the the ideas of security facility and security service in (2) we find a similar solution to be the most promising for the intelligent network. This will also tie the intelligent network to work on the OSI based systems like ODP systems.

*Definition 9 (Security Service):*  
A set of operations designed to support some aspect of security in a distributed system. (1)

The idea is that to implement security for some service we have to include the specified functionality in terms of interaction with the relevant security services assumed to exist in the basic configuration of the intelligent network. The main issue to be resolved is then *who supplies the security services?* If these services are to be used throughout the intelligent network they must be *trusted*. That is, any client must believe that these services act according to some agreed public specification, and not maliciously. Considering the number of possible customers (on the order of 108) this may seem an insurmountable problem. However, these security services are to be generic, policy independent in the sense that a very wide variety of security policy types may be implemented using these services.

*Example.* Assuming a service  $S$  is to be specified. This service may then be specified to interact with e.g. an authentication service  $X$ , for user authentication, an authorisation service  $Y$  for authorisation:

$$S \rightarrow X.\text{authenticate}(u); \quad (1)$$

which would require the user  $u$  to authenticate herself (or itself if  $u$  is a process). Similarly if authorisation with credentials  $c$  is required to access a resource  $r$ :

$$S \rightarrow Y.\text{authorise}(u, c, r); \quad (2)$$

#### 5.2 Formal definition of security policies

The ability to unambiguously decide if policy is violated or not is very desirable. This is only possible in a formal policy. Secondly a formal policy gives the possibility of consistency checks, and formal analysis. In the latter case a policy is more like a theory of formal logic. Well

known examples of such analysis includes so-called “information flow” analysis done in e.g. the Bell-LaPadula policy (3)<sup>1)</sup>.

The most far reaching reason may be the possibilities of *automatic policy enforcement*. That is, on-line continuous evaluation of policy rules on security relevant events (which are of course defined in the policy) in a system. In a sense this is of course what happens in any access control system, but only for part of the policy (the access control part).

#### Algebraic specification

Algebraic specification (see e.g. (8)) is documented to be well suited for specifying “non-functional” properties of objects like security (12). We shall consider the algebraic specification language(s) called Larch described in (10, 11), as an example. Larch specifications have theories of many-sorted first-order logic with equality as their models. As outlined above, security is basically all about compliance with some security policy. Most security policies can be modelled as first-order theories. If we include in a specification also a specification of a security policy to be enforced for that service then we stand a fair chance of having a service which will fulfill the security requirements expressed in the policy. If this is a formal specification it is possible to formally verify that an implementation is correct with respect to the specification and hence will be in accordance with the security policy which is part of the specification.

We have only hinted at the possibilities of and reasons for formal security policy models and specifications. We refer the interested reader to a mass of published work, e.g. (3, 4, 7) and in general the proceedings of the IEEE Security and Privacy workshop for recent years.

<sup>1)</sup> Let us note that the so-called Bell-La Padula model is tied to typical military security contexts. It is widely accepted that caution should be exercised if the Bell-LaPadula model is to be adapted for civil purposes.



## 6 Conclusions

We have discussed in broad terms security issues associated with intelligent networks, with emphasis on the relativistic nature of security with respect to a security policy. The principal point was to consider the intelligent network as a distributed system and use concepts from distributed systems security. The work on security in open distributed systems carries nicely over to intelligent networks. Use of the ECMA defined *security service* concept is seen as a possible way to implement security in intelligent networks. Formal specification methods should be used to enhance the quality of systems and make possible verification of security properties of services. Formal object oriented techniques are promising and algebraic specification techniques exist which should be tested. Security in intelligent network has to be taken seriously from the start. Satisfactory interdomain security solutions are critical to the successful implementation of reasonable security in intelligent networks (as it is in all open distributed systems).

We have only scratched the surface of a huge problem, and several important issues have been left out, but work will continue to solve the security problems of intelligent networks one by one.

## References

- 1 ECMA Standard 138 *Security in Open Systems, Data Elements and Service Definitions*. Geneva, 1989. (ECMA/TC32/TG9/89/.)
- 2 *Security in Open Systems A Security Framework*. Geneva, 1988. (ECMA TR/46, ECMA/TC32/TG88/.)
- 3 Bell, D E, LaPadula, L J. *Secure computer system: Unified exposition and multics interpretation*. Mitre Corporation, 1976. (Technical Report ESD-TR-75-306.) (This is the so-called Bell-LaPadula Model.)
- 4 Brewer, D F C, Nash, M J. The Chinese wall security policy. In: *Proceedings IEEE Symposium on Security and Privacy*, 1989, 206-214.
- 5 Bugge, B et al. *Methods and tools for service creation in an intelligent network*. Kjeller, Norwegian Telecom Research, 1991. (TF N34/91.)
- 6 Burrows, M, Abadi, M, Needham, R. A logic of authentication. *ACM Transactions on Computer Systems*, 8, 18-36, 1990.
- 7 Clark, D D, Wilson, D R. A comparison of commercial and military security policies. In: *Proceedings IEEE Symposium on Security and Privacy*, 1987, 184-194.
- 8 Ehrig, H, Mahr, B. *Fundamentals of Algebraic Specification 1*. Berlin, Springer-Verlag, 1985. (EATCS Monographs on Theoretical Computer Science; volume 6.)
- 9 Gaarder, K, Sneekenes, E. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal of Cryptology*, 3, 81-98, 1991.
- 10 Guttag, J V, Horning, J J, Modet, A. *Report on the larch shred language: Version 2.3*. Digital Equipment Corporation, Systems Research Center, 1990. (Technical Report TR 58.)
- 11 Guttag, J V, Horning, J J, Wing, J M. *Larch in five easy pieces*. Digital Equipment Corporation, Systems Research Center, 1985. (Technical Report TR 5.) Superseded by DEC SRC TR 58.
- 12 Wing, J M. Using larch to specify avalon/c++ objects. *IEEE Transactions on Software Engineering*, 16, 1076-1088, 1990. (Special issue on formal methods.)

# Artificial intelligence in the network: A rule based service control system prototype

BY HARALD SEIM

## Abstract

This paper describes a prototype of a service control system where object oriented techniques are combined with rule based reasoning. The background for this approach is that object oriented techniques are well suited for modelling the real world (by defining objects and properties) while heuristic rules are well suited for modelling human reasoning (by describing conditions for when actions are to be taken). In the prototype object oriented techniques are used to model the network and the subscribers, while heuristic rules are used to describe the services.

The service control system acts as a server for a telecommunication network, i.e. it receives requests and returns switching instructions. The prototype contains a graphical network simulator acting as its client. This is used for simulation and testing of services before they are introduced in the real network. The prototype may be connected to any type of network, and is today used as a service control system for an ATM broadband network at Norwegian Telecom Research.

## 1 Background

Norwegian Telecom Research considered in an early phase of its work on intelligent networks that object oriented techniques should be used. Object orientation was found to be very well suited for modelling complex systems. This because of the advantages given by classification, instantiation, encapsulation, inheritance, dynamic binding, etc.

A telecommunication network can be modelled by objects with properties and operations. Typical objects in a network are physical entities like switches, multiplexers, trunks, subscriber lines, and more abstract things like calls and connections. Some properties of a switch may be location, type, number of lines, bandwidth, etc., while its operations may be the functions for call handling.

Services are more fuzzy things to model. What you can do is to tell how you want parts of a service to work. You can for example say: "If someone calls me on this number, I would like to pay for the call." Then you add: "If someone calls me during the office hours, the call should be routed to my office, but if someone calls me in the night or in the weekend, then the call should be routed to my home." And you go on, describing different situations and conditions and the corresponding actions you want to be taken.

This situation is very typical for many kinds of services. Most services are built up by a set of actions with conditions describing when the actions shall take place. Examples of action types are: charging, routing, call completion, authorisation, encryption, etc. Examples of condition types are: time of day, day in week, origin of call, type of terminal, type of service, etc.

How should we model services allowing you to do all these kinds of cus-

tomisation, i.e. mix all kinds of conditions with all kinds of actions and action descriptions? One way is to use heuristic rules.

A heuristic rule consists of two parts: a condition part and an action part. The condition part describes when the action part may be executed. The condition part consists of a set of logical expressions (or conditions), all evaluating to true or false. If all conditions in the condition

part are true, then the actions in the action part may be executed. A common syntax for rules is: *IF conditions THEN actions.*

The rules are normally not ordered. Instead, a generic inference engine is used to determine which rule is to be fired when. When the inference engine is started, it first tries to evaluate the conditions of all the rules in the system. Then it ends up with a set of rules that

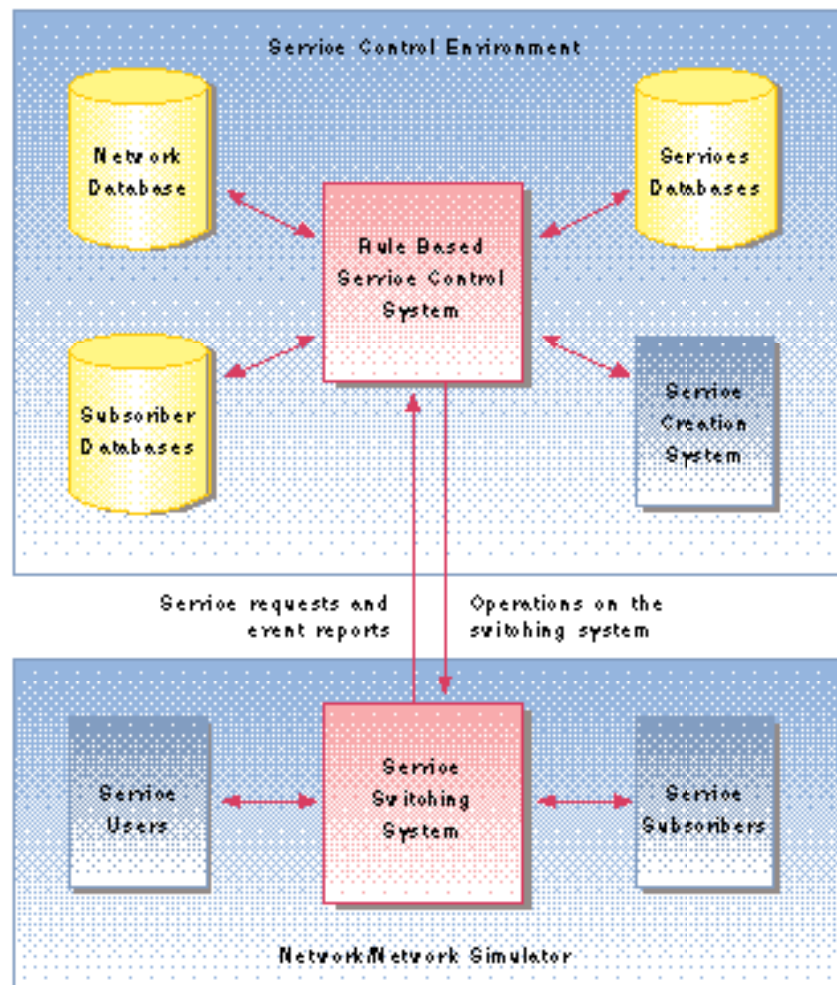


Figure 1 Overview of the rule based service control system

may be fired. If there is more than one rule in this set, there is a conflict. After a conflict resolution, the inference engine ends up with one single rule that is executed. This cycle goes on until there are no more rules to fire.

Using an inference engine saves the system developer from a lot of work with sequencing problems. New rules may be added without thinking of where and in what order. We get a very weak coupling between different rules, and this is something we can take advantage of when the system has to be modified. This is very important when dealing with frequently changing telecommunication services.

## 2 System overview

The service control system prototype is built on the concepts of object orientation and rule based reasoning. The main module consists of a generic inference engine, a set of rules describing the services and an object oriented model of the network and the subscribers/users. In

Figure 1, this module is called "Rule Based Service Control System".

The service control system interacts with the switching network over a communication link. It receives requests for service control and reports on events in the network. When the requests have been processed, switching operations in the network are called from the service control system.

During the last year the system has been developed from a plain stand-alone prototype to an on-line real-time service control point in an ATM broadband network. The service control point communicates with a connection controller in the ATM network by TCP/IP over an Ethernet connection.

For testing purposes, a graphical network simulator may be used instead of or in addition to a real switching network. Then the operations on the network are simulated and the current network status is displayed on a graphical screen.

In addition to the network, the service control system is connected to a number of databases containing information about the network, the subscribers and the services.

The network and the subscriber databases contain the object oriented models of the network and the subscribers. The network model has *access points* representing the points where the service users are connected, *legs* representing parts of connections and *connection points* representing the connection of two or more legs. All legs and connection points concerning one call, are contained within one *socket*.

The service databases are really rule bases, containing heuristic rules and some objects used for classification of services, requests and events and for storing of temporary data.

The rules are used for reasoning about the network model. Data in the network model is used by the conditions in the rules and modified by the actions. The

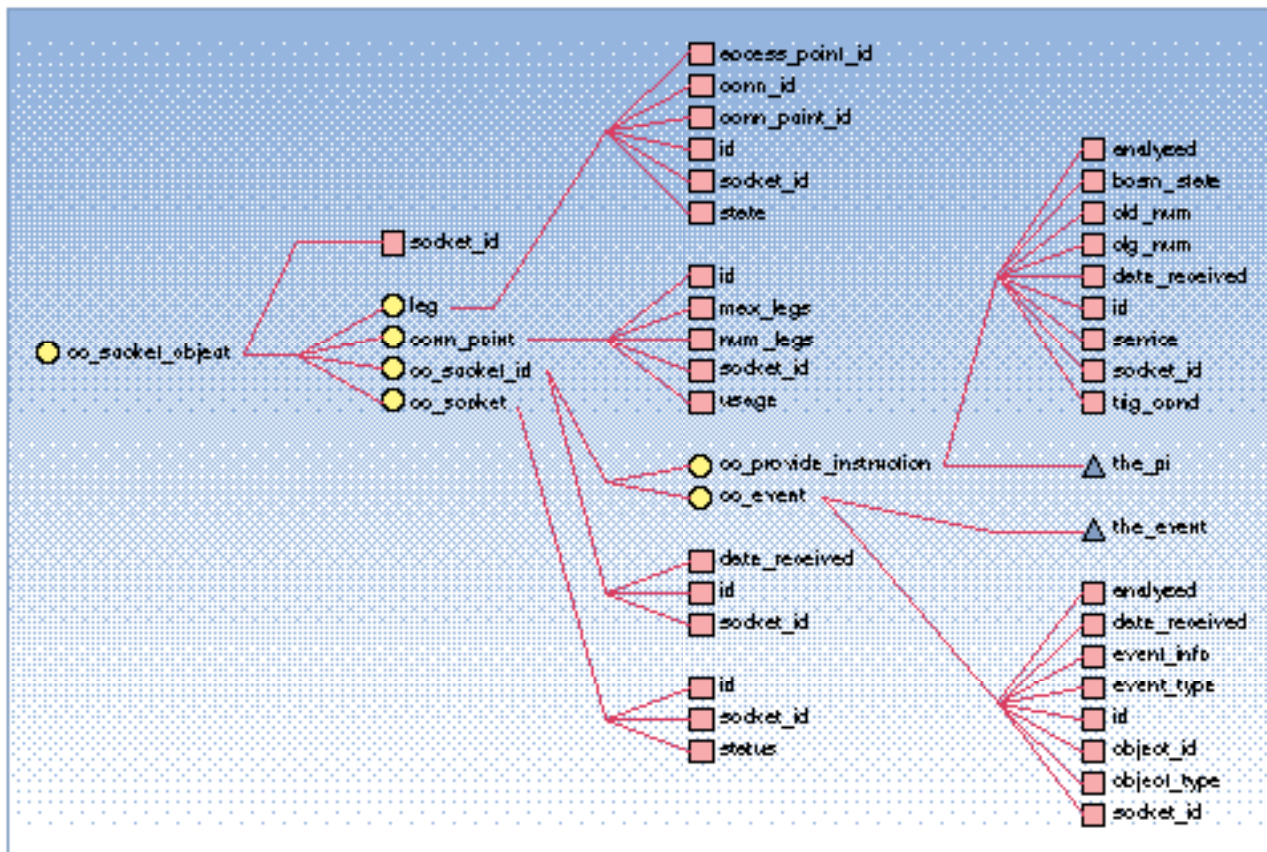


Figure 2 A part of the object hierarchy showing some of the objects used in the network model



real world is covered behind the surface of the objects, and abstracted and simplified by the object properties and operations. This makes the service control system independent of the underlying network.

Connected to the service control system is also a service creation system. This is a development environment for both the object oriented network model and the rule based service logic. The service creation system contains editors for objects and rules, graphical displays of the object hierarchy and the rule network, compiler, etc.

### 3 Object oriented network model

As mentioned above, an object oriented network model is used internally in the service control system to abstract and encapsulate the real network. The model can be considered as a map of the real world, containing only the information necessary for the service control system, e.g. only objects related to call handling.

The prototype system is based on the "Connection Control Model" drafted in "ETSI DTR/NA-6001, version 2, 14 September 1990". This model is using objects called *socket*, *leg* and *connection point* to describe connection handling in the switching network. Each object is characterised by a set of properties and operations that may be accessed by the service control system.

Figure 2 shows a part of the object hierarchy used in the network model. There are circles in front of the object classes, triangles in front of the object instances and squares in front of the object properties. Operations on the objects are not shown in the figure.

The root class in the hierarchy is called *cc\_socket\_object*. The classes *leg*, *conn\_point*, *cc\_socket\_ind* and *cc\_socket* are all children of this class. Because of this, it is possible to access instances of all these classes by referring to the root class. A property (attribute) called *socket\_id* is inherited by all objects in the hierarchy. This is used to identify object instances belonging to the same socket.

If we take a closer look at the *leg* class, we see that it has the following properties: *id* to distinguish between different leg instances, *conn\_id* to distinguish between several legs in hold state, *state* to represent the connection state of the leg (e.g. free, unjoined, joined, etc.), *socket\_id* to identify the socket it belongs to, *conn\_point\_id* to identify a possible connection point instance to which the leg is connected and *access\_point\_id* to identify a possible service access point instance in the remote end.

The possible operations on legs are: *create* to create an instance of a leg object, *free* to delete an instance of a leg object, *join* to connect a leg to a connection point, *split* to disconnect a leg from a connection point, *send\_receive* to send or receive information on a leg, *modify* to modify an attribute value in a leg and *poll* to read an attribute value in a leg.

When the service control system receives a message from the switching network, it makes an instance of either a *cc\_provide\_instruction* object or a *cc\_event* object, depending on the type of message received. The properties of this object instance are used to store the parameters contained within the received message.

During the service execution, these property values are used to analyse and classify the message. Together with the other network objects, this information is used to make operations on the network.

### 4 Rule based service logic

The service logic is contained within the rule bases connected to the system. The rules are used for analysis of network status, incoming requests and event reports before actions are taken. If more information is needed before a decision is taken, the network is polled.

The rule bases are loaded and unloaded depending on their use. When there are no rules loaded into the system to handle a particular situation, the rule bases are searched for additional rules.

One goal with the rule based service control system is to have a service independent system, i.e. a generic system where all kinds of services can be executed. In a service independent service control system there is no service specific functionality and no reference to any particular service. This means that if a new service is to be introduced, no modification of the service control system itself should be necessary.

This problem is very difficult to solve by sequential programming. Traditional software systems have *formular-driven* control, i.e. they consist of a hard coded pre-defined sequence of statements. This implies that a traditional program cannot call a routine if not a reference to that routine is hard-coded into the program before compilation.

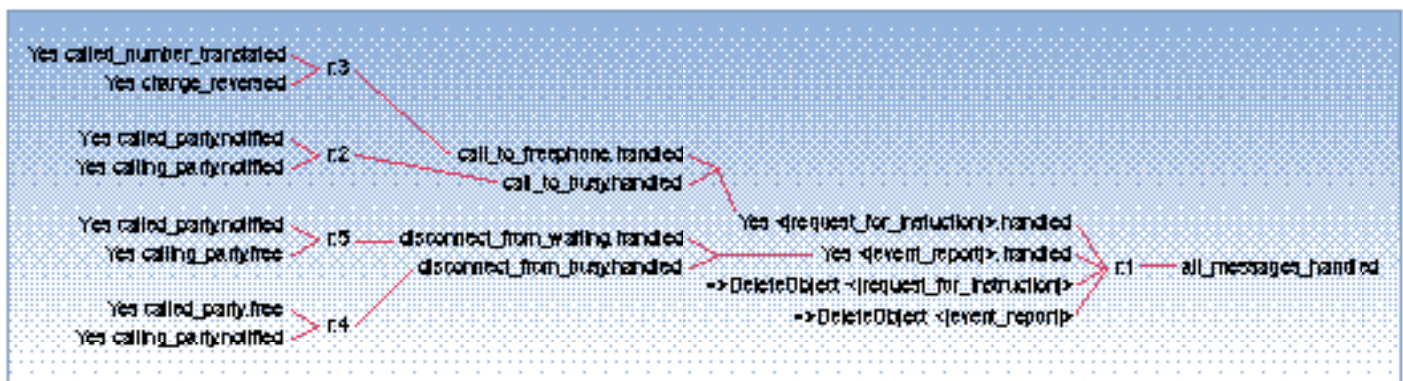


Figure 3 Example of how rules can be linked together via common data

Many object oriented systems have a capability called *dynamic binding*. This may partly solve the problem, since the procedure references may be changed at run-time. But there must still be a call to a routine, even if the routine itself is bound to the call at run-time.

Rule based systems have a much more powerful way of solving this type of problem, called *pattern-driven control*. In a rule based system there are no routines to be called. Each rule is triggered by its own conditions and independently of other rules. The connections between different rules are data rather than routine calls. The rules share a common set of data, very often an object oriented data model, and the program flow is continuously changed as the data values change.

This last technique is utilised in the prototyped system. A model of the switching network, the subscribers and the service requests are used as a common data set for the rules. The rules themselves describe how to execute services by setting conditions to the data model for when their actions are to be executed.

As a result of this, the service control system itself needs no direct references to the service specific rules to execute them. The only information the system needs about these rules, is in what rule base they are stored. For the same reason there is no need for references between different service specific rules in the system. A rule has only to know about itself and when it is to be executed.

Figure 3 shows an example of how rules can be linked together by common data. For example data used as condition in one rule can be a conclusion of another rule. The rules in the figure may be stored in different rule bases and have no direct reference to each other.

## 5 Fast service implementation

Heuristic rules may be used to describe problem solutions in a quite different way than procedures. Instead of describing a solution from the very beginning to the very end, all in the right order, you can describe a solution by "if ... then ..." rules in an arbitrary order. Rules may also be taken from different

sources and put together without thinking of the connection between them.

The use of heuristic rules is very similar to much of our own reasoning process, and this may shorten the link between a service idea and a service implementation. A service designer may have an idea about how a call waiting service should work by describing it like this:

*"If someone calls a busy subscriber, then both the calling and the called subscriber should be notified about this."*

*"If the called subscriber quits the current connection before the calling subscriber hooks on, then the calling subscriber should be notified and the called subscriber should be able to answer the new call."*

This is a very simple example of how a service may be described by heuristic rules. Figure 4 shows a simplified version of how the two rules may be implemented in the system.

For a more advanced service, the number of rules may be very high, and may also be increasing after some operational experience. While this would have been difficult to handle in a traditional system,

where the ordering of statements is very important, a rule based system would have taken the rules just as they are. The ordering is irrelevant, since this is solved by the inference engine.

The relatively simple way of putting rules into a rule based system makes it easy to prototype. With an inference engine, one can start implementing with only the key concepts. The system will be able to run with only a few number of rules, and through simulation and testing the rules may be changed and more rules added in an iterative "learning" process. With an object oriented data model, the rules may also work on symbolic data that are very self-describing and close to the real world.

## 6 Customisation of services

Customisation very often means parameterisation, but many times specialisation is desired as well. Object orientation makes it possible to specialise data by defining sub-classes and sub-objects. A rule based system also makes it possible to specialise the reasoning by adding custom specific rules. These rules do not affect the existing rules, and the

```

RULE: Rule call_to_busy_subscriber (#1)
If
    SETUP.analysed is FALSE
    And <|subscriber|.number is equal to SETUP.called_number
    And <|subscriber|.status is "Busy"
    And <<|subscriber|>>.number is equal to SETUP.calling_number
Then
    SETUP.analysed is set to TRUE
    And Create Object busy_subscriber <|subscriber|>
    And Create Object waiting_subscriber <<|subscriber|>>
    And Notify (busy_subscriber, "Call waiting")
    And Notify (waiting_subscriber, "Called subscriber busy")

RULE: Rule busy_subscriber_disconnected (#2)
If
    DISCONNECT.analysed is FALSE
    And DISCONNECT.source is equal to busy_subscriber.id
Then
    DISCONNECT.analysed is set to TRUE
    And Disconnect (busy_subscriber)
    And Send (busy_subscriber, SETUP)
    And Notify (waiting_subscriber, "SETUP sent to called subscriber")
  
```

Figure 4 Two simplified rules for handling of call waiting

conditions in the customised rules can guarantee that they will only work for a particular user or a particular group of users.

## 7 Operation and maintenance

In an intelligent network the service software has to change frequently to fulfill the subscriber requirements, and new service software has to be distributed in the network continuously. The service control systems should therefore have the ability to load new software and unload old software without system stop. Many rule based systems allow new rules and objects to be loaded and unloaded at run-time, and some systems allow existing rules and objects to be modified at run-time as well. It is the modularity of rule based systems that makes this possible. All rules are independent of each other and only connected via a common data model.

The prototyped system has a very flexible inference engine where adding, deleting and modification of all system parts is possible at run-time. This may be done by external routines or by loading new software from the databases.

Software maintenance is very dependent of the ease of understanding the code and the possibility to make changes without unpredictable side effects. Rules are easier to maintain than traditional code. Not only because of the syntax, but also because a rule based system is able to explain its own reasoning process. It is very helpful to understand how the system is reasoning, what conditions are satisfied and how conclusions are reached to maintain the software.

## 8 Service simulation

The prototype is offering a graphical representation of the network situation, so that the service designer can see the effect of the service logic on the switching system. Figure 5 shows a typical screen image during a service simulation. The largest rectangle is a display of the switching situation. It shows a connection point connecting two legs (between access point 2 and 10). In addition there is a leg not connected yet (from access point 16).

The three small displays show the status at the three access points. They also give the user the ability to send and receive messages to and from the network. Several calls can be handled simul-

taneously, and more access point displays may be opened if necessary.

It is possible to look at the network model at any time during a service simulation. Figure 6 shows some of the objects created during the simulation example shown in Figure 5. The first object is the provide instruction request, then follow the socket, connection point and the three legs. The (+) in front of the object names means that they are dynamic objects that will be deleted when the call is finished.

A number of telecommunication services have been developed for the rule based service control system. Examples of services that are running on the system today are:

- Call set-up / release
- Call waiting
- Call hold / retrieve
- Freephone
- Time dependent routing
- Origin dependent routing.

Common service rules are stored in the rule bases, one for each service. Subscriber specific information is stored in the subscriber databases.

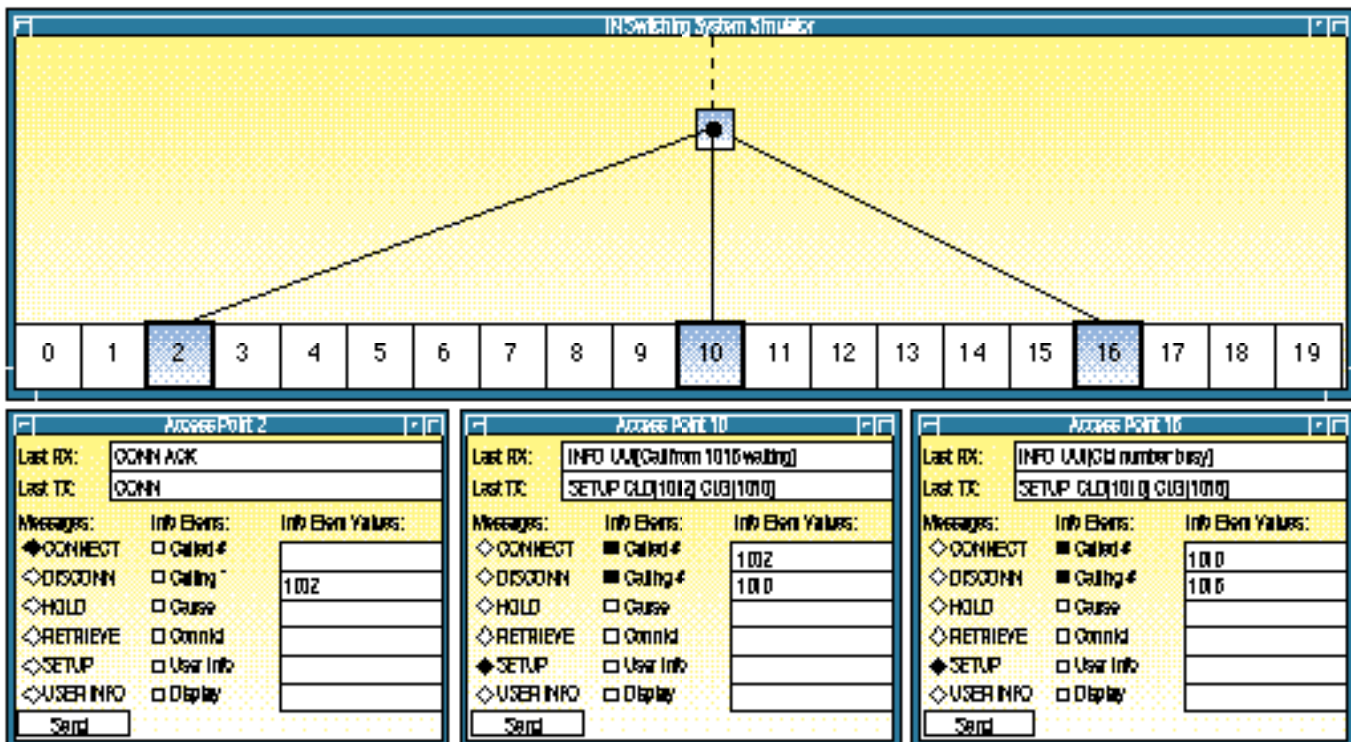


Figure 5 A typical screen image during simulation of a service on a workstation. The simulator window is displaying the switching status during a call waiting service, while the three access point windows are displaying the current status at the access points.



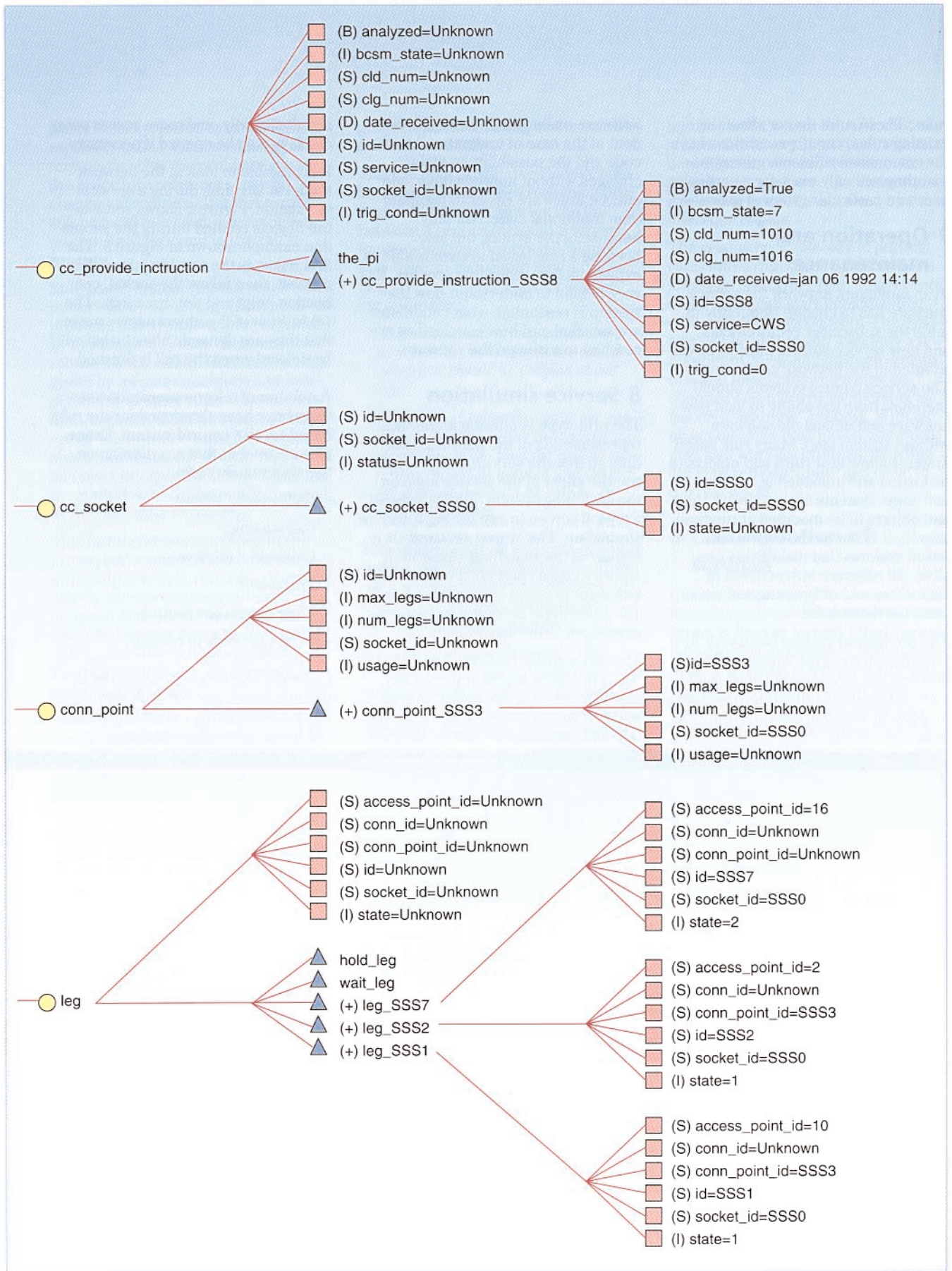


Figure 6 Some objects created in the service control system prototype during execution of a call waiting service

# Universal Personal Telecommunication and intelligent network architecture

BY JAN AUDESTAD AND BØRGE JACOBSEN

## Abstract

This paper presents an object oriented (OO) approach to UPT architecture allowing UPT users to roam between networks of different type and ownership. The architecture is based on a recursive definition of objects, i.e. where objects can be attributes of objects. This allows services to be viewed at different levels of granularity and designs where the actual software is distributed in an arbitrary manner between the machines making up the physical architecture of the network.

621.395:621.396.931  
621.39.05  
681.324

## 1 Mobility

It is convenient to distinguish between three types of mobility forming a bottom-up hierarchy as follows:

- terminal mobility
- terminal portability
- personal mobility.

*Terminal mobility* is the type of mobility offered by land or satellite mobile systems and can be regarded as geographical distribution of the subscriber line. The benefit for the user is that the network can be accessed while moving and independently of fixed subscriber line access points.

*Terminal portability* means that a user terminal can be plugged in at an arbitrary access point which may be at a fixed installation or at a mobile station. The benefit for the user is that user specific capabilities of the terminal can be exploited at all network access points.

*Personal mobility* implies that the user can initiate and receive calls on the basis of a single personal number at any fixed, portable or mobile user terminal. The

three levels of mobility and their relationships are illustrated in Figure 1.

Personal mobility thus makes use of all levels of mobility. Personal mobility is the core element of Universal Personal Telecommunications (UPT). UPT will become one of the basic service capabilities of future networks and will therefore require that all future network architectures support aspects of personal mobility (roaming, secure network access, uniform access procedures, etc.).

## 2 UPT requirements

UPT will impose several requirements on future networks. Each UPT user should ideally have a single UPT number by which he or she can be reached at any network access point. On this number it should also be possible with different user roles, e.g. the user as a private person or as employee. The services available for the user may be different for different roles. As employee the services may not only be personal but also contain capabilities subscribed to by the company. This implies that service profiles can be distributed in the network depending on role. However, there may also be

cases where it is more convenient with separate UPT numbers for each role. This should be subject to choice by the user and not be a restriction imposed by the network.

The network should be capable of providing the user with accustomed or user specific user-network access procedures at all locations. Examples of user specific procedures are customised alerting, customised announcements or text and abbreviated number/hot-line selection. It should be possible for the user to roam between networks owned by different network operators. This capability should include roaming between different LANs and VPNs.

The UPT service should not depend on network type, e.g. it should be possible to roam between ISDNs, dedicated data networks and B-ISDNs. This capability implies that a common service control should be defined for all network types. Note that if a shared numbering plan does not exist between these networks, then the user must have one UPT number for each network.

All of the above requirements have impact on the architecture. It is

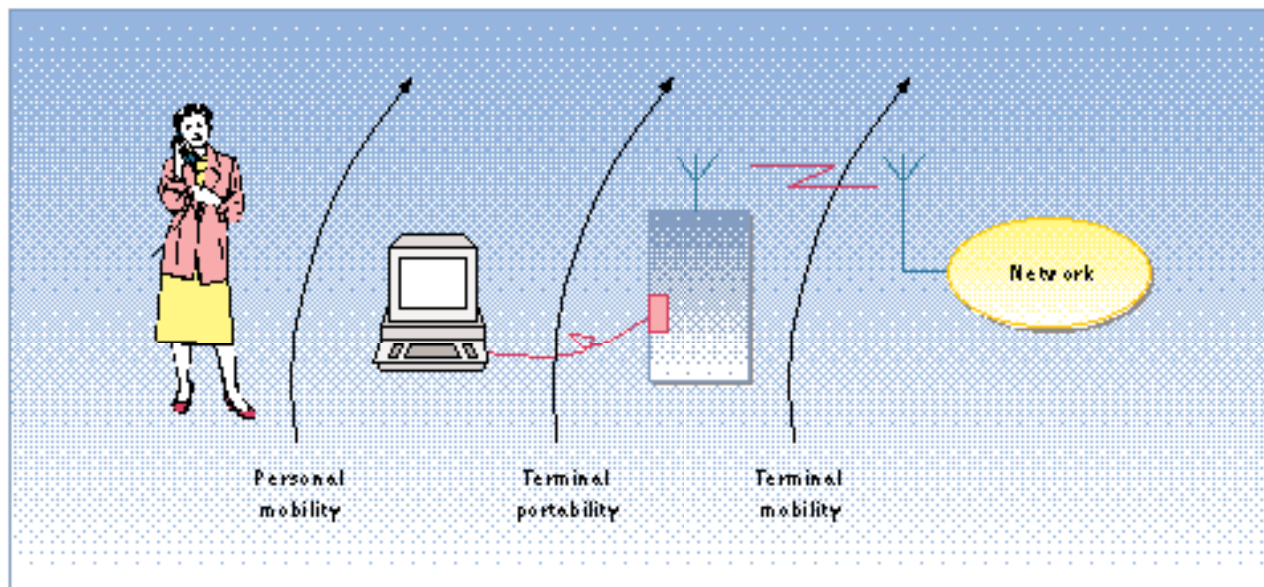


Figure 1 Mobility hierarchy



particularly evident that the architecture should not be optimised in respect to one network design only.

### 3 Principles for UPT architecture

A clear distinction should be made between conceptual architecture and physical realisation. The conceptual architecture should be independent of physical realisation. The requirements listed in section 2 above showed that this will in particular be true for UPT.

A conceptual architecture (e.g. like the one defined formally in CCITT recommendation X.407) can be used to show aspects of the system from different angles and at different level of granularity. The different levels of granularity are

obtained by refinement of less detailed views in a recursive manner. (1)

A general top level conceptual architecture of a telecommunications network is shown in Figure 2. It consists of three interacting domains each representing a distributed process.

Management domain responsible for service creation, service subscription management, service deployment and service execution management.

Service domain responsible for execution of the telecommunication service, including manipulation on the switching functions of the connection domain.

Connection domain representing transmission and switching functions.

A refinement of the general architecture valid for UPT call processing is shown in Figure 3. The service domain has been sub-divided into an A-party service and a B-party service. The connection domain has been refined as access points and switched path. Each of the new elements may represent a distributed process. The manipulation of the connection domain, i.e. the switched path, is performed by either the A-party service, the B-party service, or both.

This simple call processing architecture may be mapped onto its realisation architecture, or for that matter, another conceptual architecture (for IN it is not clear whether the current architecture is conceptual or reflects realisation). An example is shown in Figure 4 where the A-party side is mapped onto an IN and

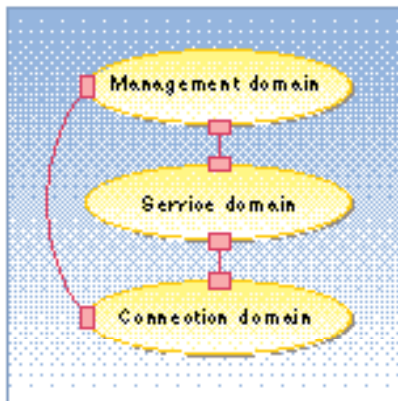


Figure 2 Conceptual architecture of a telecommunications network

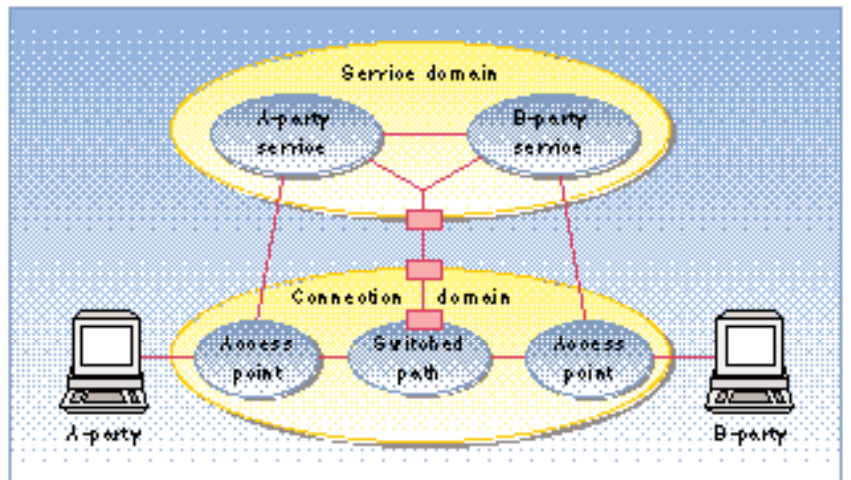


Figure 3 Refinement of the general architecture

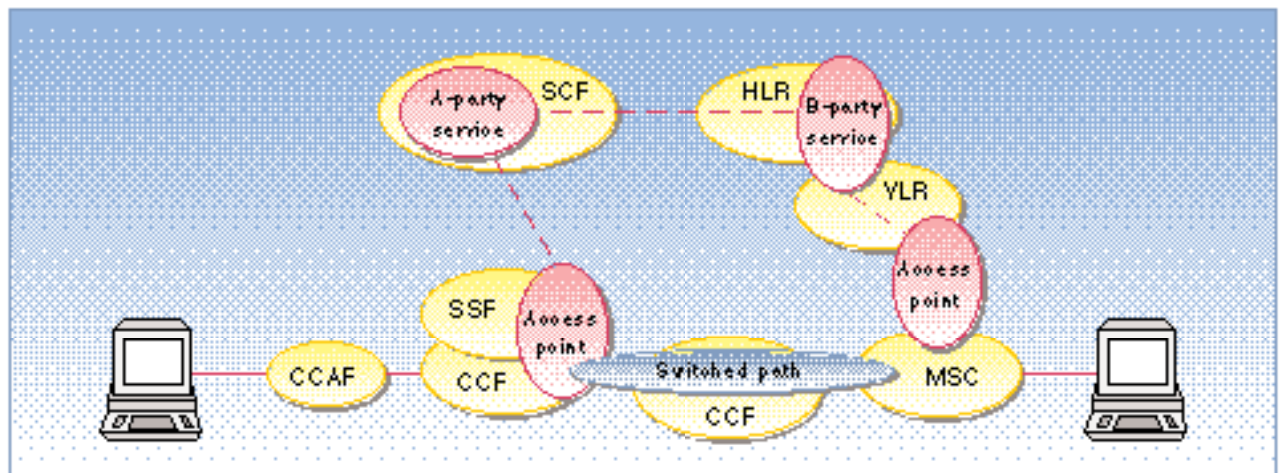


Figure 4 Mapping the conceptual UPT architecture onto IN and GSM



the B-party side is mapped onto the GSM system. There are two important points in this example. First, it is possible to define a mapping in each case and allocate the functions of the conceptual architecture to functional elements in each realisation. Second, each function of the conceptual architecture may map onto a single functional element or be distributed over several elements. It should also be noted that the functions may be realised in independent ways in IN and GSM. The only requirements is that the functions A/B-party service, access point, etc., behave according to a common UPT specification. The conceptual architecture thus meets the requirements of mappability referred to in section 2.

#### 4 Object oriented approach to UPT

Object Orientation (OO) is a powerful method for specification and implementation of complex systems. Some important aspects of OO are: (2)

- Modularity
- Reusability
- Abstraction
- Inheritance
- Information hiding
- Encapsulation
- Dynamic binding.

The UPT service will be complex. The distributed nature of the service adds additional complexity. It is important to find a suitable method for handling this.

Figure 5 shows a UPT call service modelled as interacting objects. The attributes of an object may be objects themselves, but this structure of the service object is hidden at the higher levels of abstraction. The objects shown in Figure 5 are in the service domain but since services generally imply manipulation of connections, some of these objects must manipulate the connection domain. At the given level of abstraction this is hidden within one or more of the objects.

Figure 5 shows the service execution. It does not consider physical distribution of the service but focuses on the service modelling.

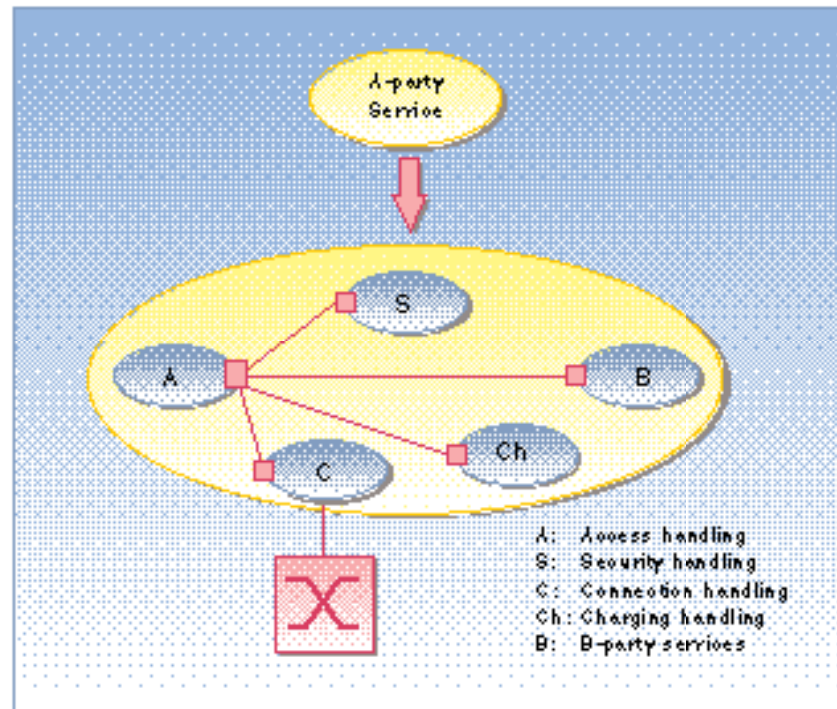


Figure 5 UPT service execution architecture from an object oriented view

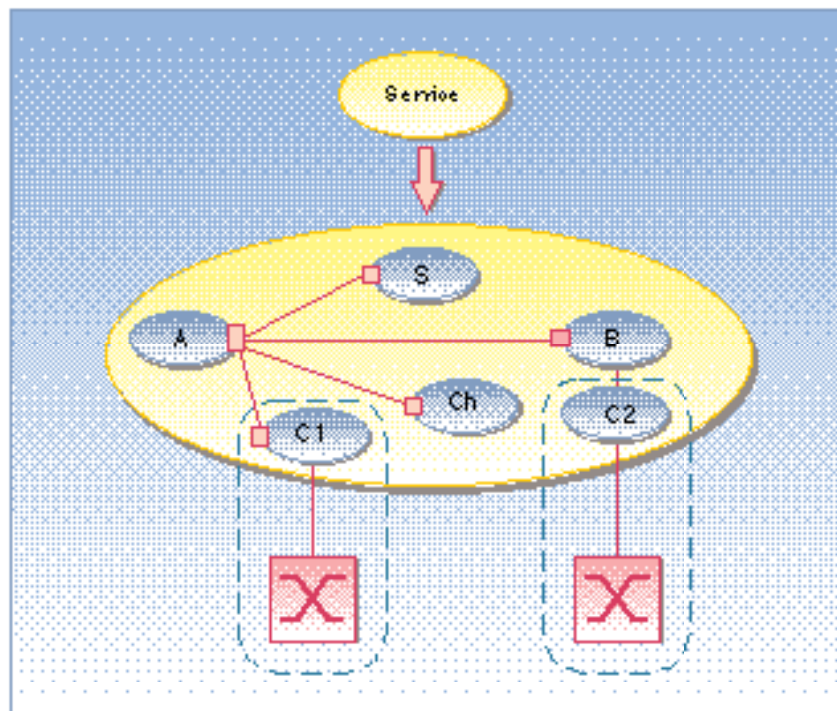


Figure 6 Object oriented mapping of UPT service onto physical networks

The A-party service objects are invoked to initiate the UPT service. One of its attributes is the access handling object (A). The access handling objects may have other objects, e.g. identification objects, screening objects, as attributes.

Object A invokes a security object (S), a charging object (Ch), a connection handling object (C) and a B-party service object (B) where C manipulates the switches and B handles the B-party services. The B object may be sub-divided

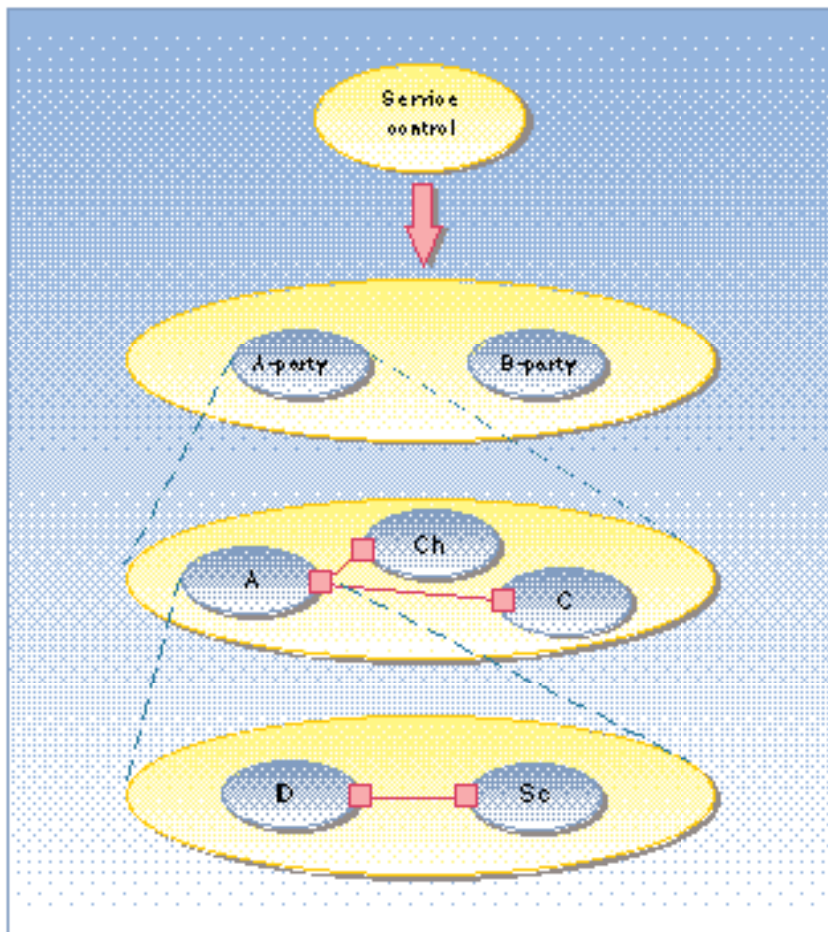


Figure 7 Object oriented decomposition of a service control object

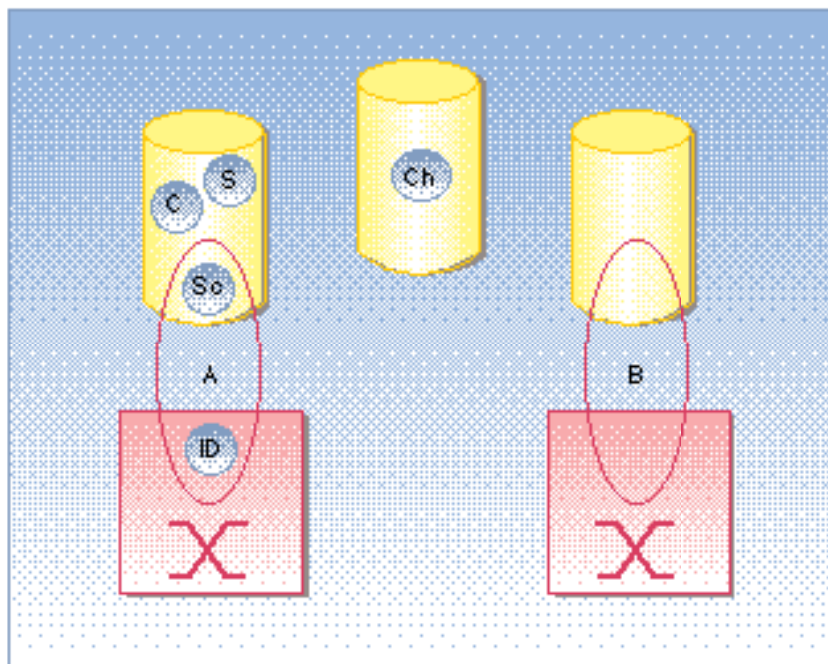


Figure 8 Distribution of objects in physical realisation

further to reflect the constituents of the B-party service. This sub-division is not considered here.

Figure 4 showed a mapping of services onto an IN system and a GSM system. Figure 6 contains a corresponding mapping for the OO approach. In the example two connection objects (C1 and C2) are required in order to manipulate the switches. The objects C1 and C2 have inherited their general characteristics from a superclass C. The adaption to the actual networks is hidden inside the objects so that a separate connection object sub-class may be defined for each type of network. In this view we can regard the subscriber access lines to be contained in objects A and B, and the physical switches and transmission network to be contained in objects C1 and C2. Thus, peculiarities within each specific network are hidden from the other objects which constitute the service. The objects of type A or type B should not see any difference between objects of type C1 and type C2. The same operations are performed on both but their internal actions are performed on different types of networks. The type of object to be invoked must thus be chosen at the time of invocation. In object oriented design this is possible by dynamic binding. Communication between the objects will use available protocols in the networks. Though C1 and C2 look the same for A- or B-type objects, the protocols chosen for communication between them may not be the same. From the service point of view, only the logical information flow is important. The objects can be distributed on several different nodes in the networks which offer the service. In the short term this distribution may be done manually. In the long term the run-time system should optimise the actual distribution automatically. The encapsulation and modularity offered by OO will simplify the distribution task.

The object oriented approach yields an architecture which is easy to decompose into different levels of granularity. Figure 7 shows the decomposition of a service control object into gradually finer parts. The top level consists of the service control object only. This object is then decomposed into types A, B, C, S, and Ch. The next level of decomposition is

only shown for object A which consists of an identification object (ID) and a security object (Sc).

In a physical realisation the objects may be distributed in such a way that the various attribute objects it may contain are instantiated at different machines. This is illustrated in Figure 8 for the decomposition in Figure 7. In this example the A-service objects are distributed on three machines and its contained access object (A) is again distributed on two of them.

This capability of distributing objects is particularly important for UPT since it allows objects to be moved and instantiated at different places as the user roams in the network. In this way optimum use of resources and the best possible service performance can be planned into the service.

The difference between model and implementation is very distinct in this architecture. The model contains classes and the implementation is the actual instantiation of these classes, i.e. the objects. In the current IN architecture (3) as used by ETSI and CCITT, the distinction may not be so clear.

## 5 Conclusion

The object oriented approach to the UPT architecture removes unnecessary and undesired bindings of the UPT service to physical realisation. The decomposition of the service into interacting objects is independent of network design, network ownership and network type. The decomposition depends only on the particular UPT service offered to each individual user. The mapping of this decomposition onto network resources can be done in an optimum way maintaining all service features independently of this mapping.

In this paper only a simple two-party call service has been analysed. The same approach will apply to more complex services and to other aspects of the UPT service such as registration and alteration of subscriber parameters.

## References

- 1 CCITT X.407. Message handling systems: Abstract service definition connections: recommendations X.407. In: *CCITT Blue Book*, vol VIII, fascicle VIII. 7, 200-227.
- 2 Meyer, B. *Object-oriented Software Construction*. Englewood Cliffs, N-J., Prentice-Hall, 1988.
- 3 *CCITT Q.1200-Series of Recommendations*. Geneva, 1992.