GMD Report 23

GMD –
Forschungszentrum
Informationstechnik
GmbH

Jiri Kuthan

Internetwide Network Computing - Client Side

June 1998

GMD –
Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin
Germany
Telefon +49 -2241 -14 -0
Telefax +49 -2241 -14 -2618
http://www.gmd.de

Anschrift des Verfassers/Address of the author:
Jiri Kuthan
Institut für Offene Kommunikationssysteme
GMD – Forschungszentrum Informationstechnik GmbH
Kaiserin-Augusta-Allee 31
D-10589 Berlin
E-mail: Jiri.Kuthan@gmd.de

ISSN 1435-2702

Abstract

This work concentrates on the developing *Network Computing* technology. The new FOKUS-developed approach based on the *Internet Service Management Protocol* (ISMP) is introduced. This approach to the automatization of accessing services over the Internet extends the contemporary intranet NC reference model to an internetwide scale, accomplishes the administration of NCs and provides users with increased mobility.

The concept and technical issues of the *client-side* are the subject of this work. System architecture is introduced, ISMP is defined and analyzed, and technical solutions are discussed on a more detailed level. Final chapters cover a prototypical Java implementation.

Keywords

network computing, service management, mobility, electronic service subscription

Acknowledgments

This Master's thesis, being submitted to the University of Salzburg's Computer Science Department, is the result of work done by myself at the GMD under the direct supervision of Prof. Dr. Hofmann from the University of Salzburg and Lutz Henckel from GMD-Fokus.

I would like to pay special thanks to all who have introduced me to the wonderful world of the Internet, supported my thesis, collaborated with me on our project and provided me with many helpful hints (alphabetically ordered):

Thomas Baumgart, GMD-FOKUS

Prof. Dr. Hornst Clausen, University of Salzburg

Lutz Henckel, GMD-FOKUS

Prof. Dr. Ulrich Hofman, University of Salzburg

Joerg Schilling, GMD-FOKUS

Dr. Henning Schulzrinne, University of Columbia

Stephan Waßerroth, GMD-FOKUS

I would also like to thank Rachel Rückeis (TU Berlin) for linguistic correction of my thesis.

Prerequisites

Readers of this document should be familiar with the fundamentals of operating systems and technical aspects of the Internet. Knowledge of the Java-language and Bourne-shell is necessary only for understanding the prototypical implementation.

Typographical Conventions

Definitions and terms important for further reading are printed in italics.

Example:

These hazards are minimized by the following *ISMP reliability means*:

References are enclosed in brackets.

Example:

The reliable transport-level protocol in the Internet protocol stack is TCP [rfc793] or its transaction-oriented extension T/TCP [rfc1379, rfc1644].

In order to provide curious readers with source code references, the relevant pieces of code are referred to in footnotes.

Example:

ISMP is a *3-way handshake protocol*[1]

1. Implemented in ISMPoverTCP.makeTransaction().

Table of Contents

Figures

Tables

1 Introduction

1.1 The General Concept of Network Computing

Nowadays, the client-server architecture is the most common architecture used in distributed computing systems. Typically, PCs working as clients are connected to full-powered UNIX-based servers. This architecture has become extremely popular and has replaced the formerly used mainframe model almost completely — it is fast, scalable and reliable.

On the other hand, solutions based on this architecture cause higher expenses — according to a study by the Gartner Group [gg], the cost of ownership (including initial cost, hardware maintenance and software administration) of a client (called a *fat client*) reaches up to 12 thousand US dollars a year which is definitely not a negligible amount.

The idea of *Network Computing* is to replace the fat clients with so called *Network Computers (NC)* — affordable, easy-to-use devices, administrated and maintained by a network. The NC operating system, applications and user-data are centrally stored and downloaded via the network to the "stateless" NC as needed (therefore it makes no difference which client machine a user logs in from). Much of the application processing takes place on the NC. The yearly costs of such a client would sink down to $2,500 — nearly an 80% reduction!

In May 1996 a consortium of computer industry leaders: Apple, IBM, Oracle, Sun and Netscape, announced a set of guidelines for developing the thin clients. This architecturally neutral set of guidelines, called "NC Reference Profile 1" [ncrp], provides a common set of standard features and functions across a broad range of scalable NCs. This reference profiles especially specifies the minimum hardware requirements of the NC, supported Internet protocols, multimedia formats and security features.

The first NC implementations of this reference profile are available on the market.

1.2 Benefits of Internetwide Network Computing

The current NC model requires manual administration of the thin clients over a local network.

The idea of the FOKUS researchers is to supply this model with an additional protocol (*Internet Service Management Protocol - ISMP)*, which will be used to automatize subscription, registration, authorization and administration. Users will just rent any network service they need: E-mail, News, applications, etc. by using this protocol.

This enhanced model has the following advantages:

- The protocol supports mobility — NC users can access all services from any NC around the

world (“subscribe once and use anywhere”).
- Users do not have to worry about administration and maintenance at all — they just order and pay for services. In addition, even most of the network administrator’s job will be done automatically.
- Users can feel free to choose any service provider they want.
- The ISMP protocol relies on the underlying TCP/IP protocol stack, which means that the thin clients can be connected to their servers over any LAN/WAN network supporting TCP/IP.

1.3 FOKUS-Architecture of Network Computing and the ISMP Protocol

1.3.1 Architecture Requirements

The implementation of the internetwide network computing concept shall meet the following requirements:

- NC customers can decide to subscribe/unsubscribe any available service provided by any provider at any time (e.g. a user cancels a subscription to a disk-service offered by a provider P1 and subscribes to P2, if he or she finds out that a provider P2 provides the same service for half the price).
- Providing services will be fully transparent — all server system changes on the provider-side shall be negotiated over the ISMP without any user’s manual intervention. (E.g. a crashed file server would be automatically replaced by a backup server without any manual reconfiguration of the NC).
- The usage of subscribed services will be protected by authorization means.
- An account of service usage will be kept. In particular, an account of login and subscription times, type and extent of used resources will be kept.
- NC users will be able to use NCs in remote regions without noticing any differences to local usage — their environment and subscribed services will be available over the whole Internet.
- Scalability. The architecture will be able to serve high number of clients (up to millions).

1.3.2 Architecture Components

The architecture consists of the following fundamental components (Figure 1):

- ***Network Computer (NC)*** - low budget computer (ca. 600 USD) consisting of a CPU, graphical adaptor, sound adaptor, network adaptor, monitor, keyboard and mouse. The NC is also equipped with a low-capacity hard disk for caching the downloaded operating system, WWW, etc. A user-friendly graphical interface for registering for services at various providers as well as software for using the subscribed network services (e.g. Netscape Communicator for accessing WWW/E-mail/news) is preinstalled.

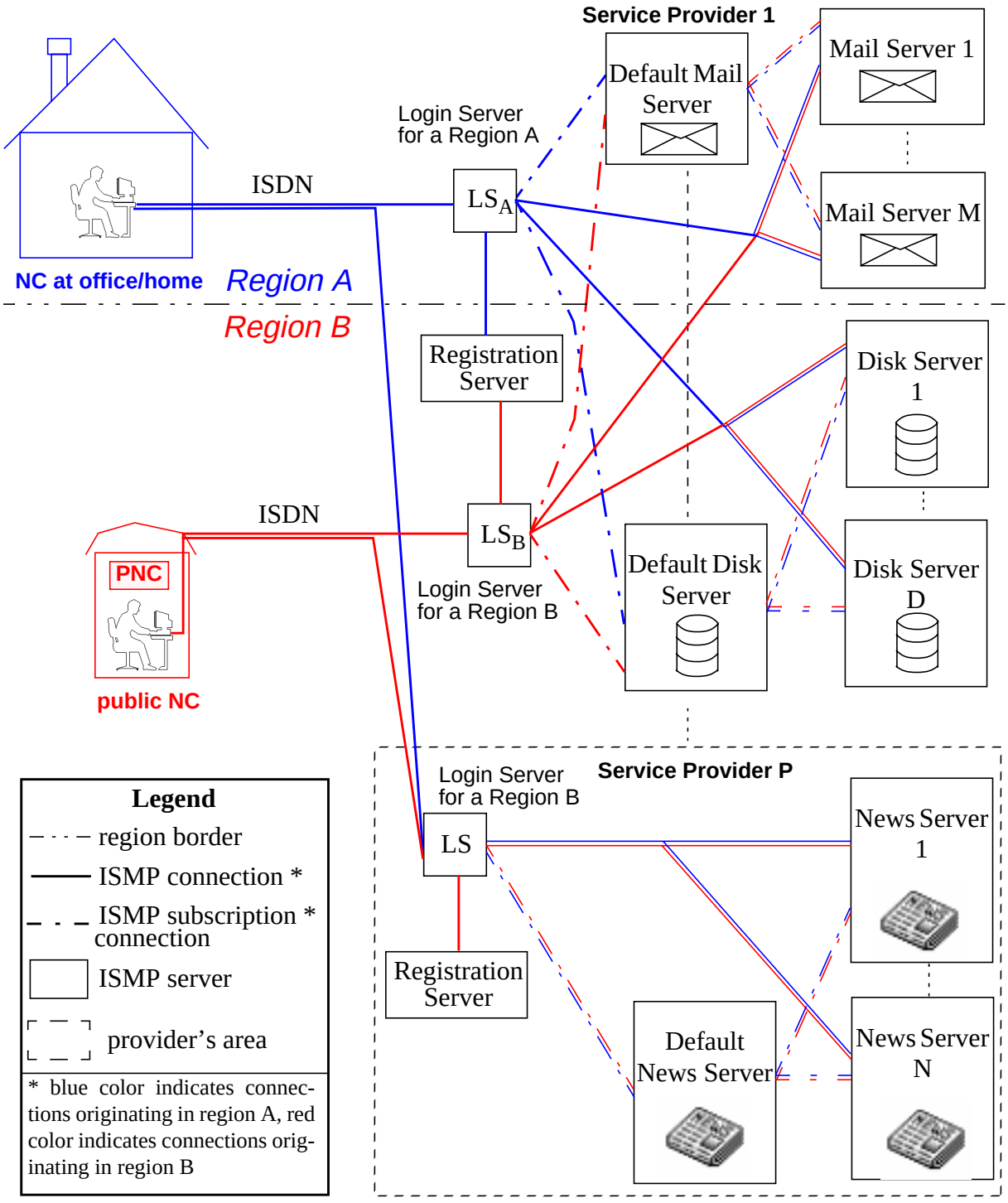

Figure 1: NC-Architecture

- ***NC Session Manager (NCSM)*** - is the software responsible for controlling the NC, subscribing, unsubscribing and the managing network services (disk, news, E-mail and various applications) provided by the service providers for the NC user. The NCSM has a simple-to-use graphical user interface; it uses the Internet Services Management Protocol in order to

perform all management functions automatically.

- ***Login Server (LS)*** - controls access to a provider's services; it checks user-requests against a database of all registered users (RS). The LS is the only gateway for NCs — all specific requests are forwarded over the LS to their specific destinations upon the request's service.
- ***Registration Server (RS)*** - deploys a distributed[1] database of users registered at a provider.
- ***Default Service Server (DSS)*** - controls access to a specific service. The DSS performs load balancing on service subscription — it selects the least loaded *Service Server (SS)* before assigning it to an NC. The assigned server may be exchanged by administrators in case of a failure for instance.
- ***Service Server (SS)*** - makes a specific service available for users. (E.g. in case disk space is required by a user, a new directory is created and exported to the requesting NC). The available services are used over common Internet protocols (s. [ncrp]). The NC never knows in advance which SS it will be dynamically assigned to by the ISMP which makes server exchange (e.g. in case of a failure) easy and transparent for the NC.

  Currently, *mail-service* (over SMTP & POP3), *news-service* (NNTP), *WWW-service* (over HTTP), *disk-service* (over NFS) and *applications-service* (Java applets over HTTP and X Window applications) are supported.
- ***Internet Service Management Protocol (ISMP)*** - a TCP/IP based protocol supporting automatic management of the subscription to services; this protocol runs on the public side between the NC and the LS, as well as on provider's private side between the LS, DSS and SS. Its main task is electronic service subscription and access control. The protocol primitives are: *service subscription, service unsubscription, service login, service logout* and *change of service parameters*.

1.3.3 Underlying Technologies

The concept relies on many underlying technologies.

Most of these technologies are already specified in the NC reference profile [ncrp]. The architecturally neutral reference profile is intended to be an open standard specifying minimum hardware and software requirements. In particular, support for the *TCP/IP* based protocol family and for the *Java runtime environment* is required in order to provide maximum platform independence.

> *Note: The TCP/IP protocol stack is the architecture independent protocol family behind today's Internet. It supports the most widely used services, such as E-mail, WWW, file transfer, etc. For more information, see the "Internetworking" section in the bibliography.*

1. The database can be distributed over NIS, NFS or even completely different DB-system.

Note: "Java™ is a simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language." ([jlwp]). For more information, see the "Java" section in the bibliography.

Additional used technologies are implementation-specific and are used within the FOKUS NC project. They are described in the following paragraphs.

Netscape Communicator has been selected for accessing WWW, mail, news and Java applications — it is today's most popular software and is used by most of the Internet community. Moreover, it is available on many platforms.

In order to provide NC-users with satisfactory data transmission speed, *ISDN* (64 kbps) is used for this prototype. Analog lines would be too slow and higher-rate lines (e.g. ATM) are expensive and still not commonly available. Other technologies may be used in the future if they become available and guarantee satisfactory bandwidth (e.g. *DSL*).

A *PC* with *Linux-OS* has been selected as the hardware & OS platform for evaluation purposes — all required underlying technologies (Java, Netscape, TCP/IP protocol stack, etc.) are implemented on this platform. Porting software written for this platform is easy, because most of the NCSM code is written in Java. All platform-dependent features are separated into Bourne-shell scripts which are easy to port to other UNIX-like operating systems.

The following PC peripheries are required: an ISDN adaptor, a VGA adaptor and monitor, a low-capacity hard disk for caching network data and the most recent version of the OS, keyboard and mouse. In the future, low-price custom-motherboards with a Java-chip may be available.

1.4 An Example of Usage

This simple example demonstrates how an NC can be used.

- A customer buys an NC with a preinstalled NC-OS and connects it to an ISDN socket at home.
- The customer dials the telephone number of a service provider and registers with him. A directory with configuration files for keeping track of subscribed services is assigned to the customer. If the version of the operating system which is installed on the NC is obsolete, the newest version will be downloaded automatically.
- The customer subscribes to various services from various providers: E-mail, a 100 MB disk and a tax-processing application. He or she uses the application to calculate their taxes and saves the tables on the rented disk.
- When the customer goes on a journey all his services are available. He or she can log-in at a public NC at a railway-station, read incoming E-mails or send his previously calculated tax-

declaration to the tax-authorities. Additionally, the customer might want to lower the size of his subscribed disk for the length of time he is on journey in order to lower the subscription costs.

2 The Internet Service Management Protocol

The Internet Service Management Protocol (ISMP) is a TCP-based protocol for automatized managing services provided over the Internet according to the architecture described in the introductory chapter. Its main task is electronic service subscription and access control. This chapter defines its format and functions and examines its reliability means.

2.1 The ISMP Primitives

To support automatic negotiation of service-usage five protocol requests are defined in the protocol. They can be answered by the server with either a positive response or with an error code.

Table 1: protocol request types

request type	description
subscribe to a service	• A user is added to the database of service users and service-specific resources are reserved in response to this request. • Only users known by a provider may subscribe. If they are unknown they must identify themselves at the provider to receive a provider-unique username and password. • The subscription to the service remains until it is explicitly unsubscribed.
log in to a service	• This request is sent to the LS if the user wants to use a subscribed service. • If the service is currently not available (e.g. if the login request was sent immediately after the subscription request and the server has not made the service available yet) the user receives a negative response, otherwise he can start using the service.
log out from a service	• In response to this request the server stops a service provided to the user.
unsubscribe a service	• In response to this request service-specific resources allocated for the user are disposed of and the user's entry is deleted from the service database.
change service parameters	• This request is used for the modification of service-specific parameters, e.g. amount of rented disk space, service password, etc. • The request is allowed only for logged-out services (for consistency reasons, e.g. changing the size of a rented disk could cause the exchange of a disk-server, which can not be done if the service is logged-in). This constraint might be removed in future protocol versions.

Note: Also registering, cancelling a registration and updating user information at a provider is mapped to these primitives. A special "provider" pseudoservice is defined for this purpose. The "login" and "logout" primitives are not defined for it.

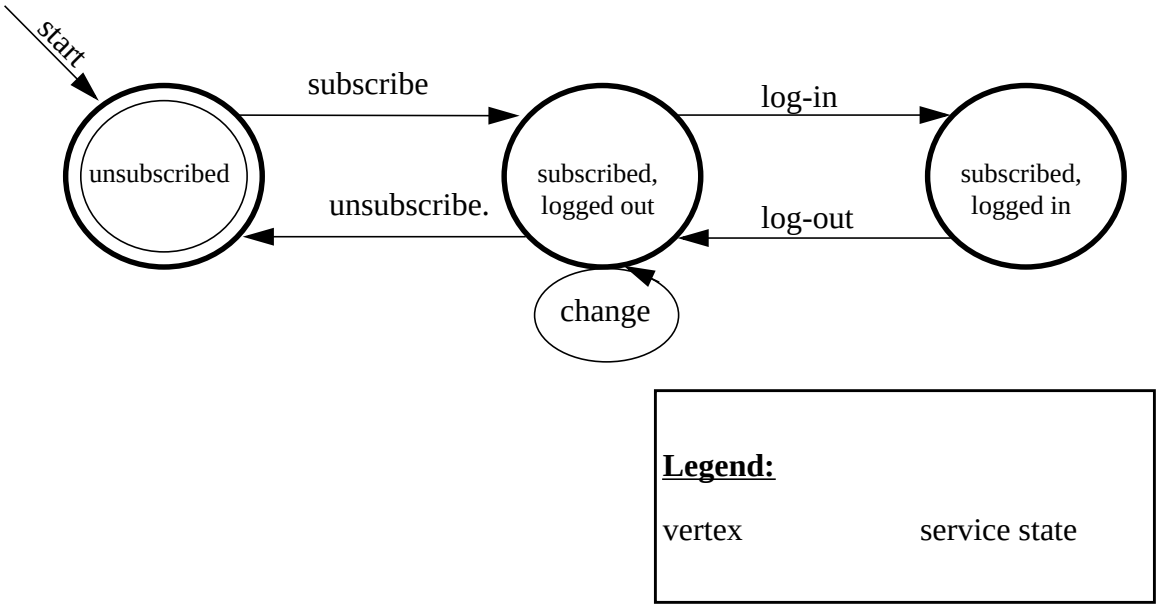

Figure 2: Service State Diagram

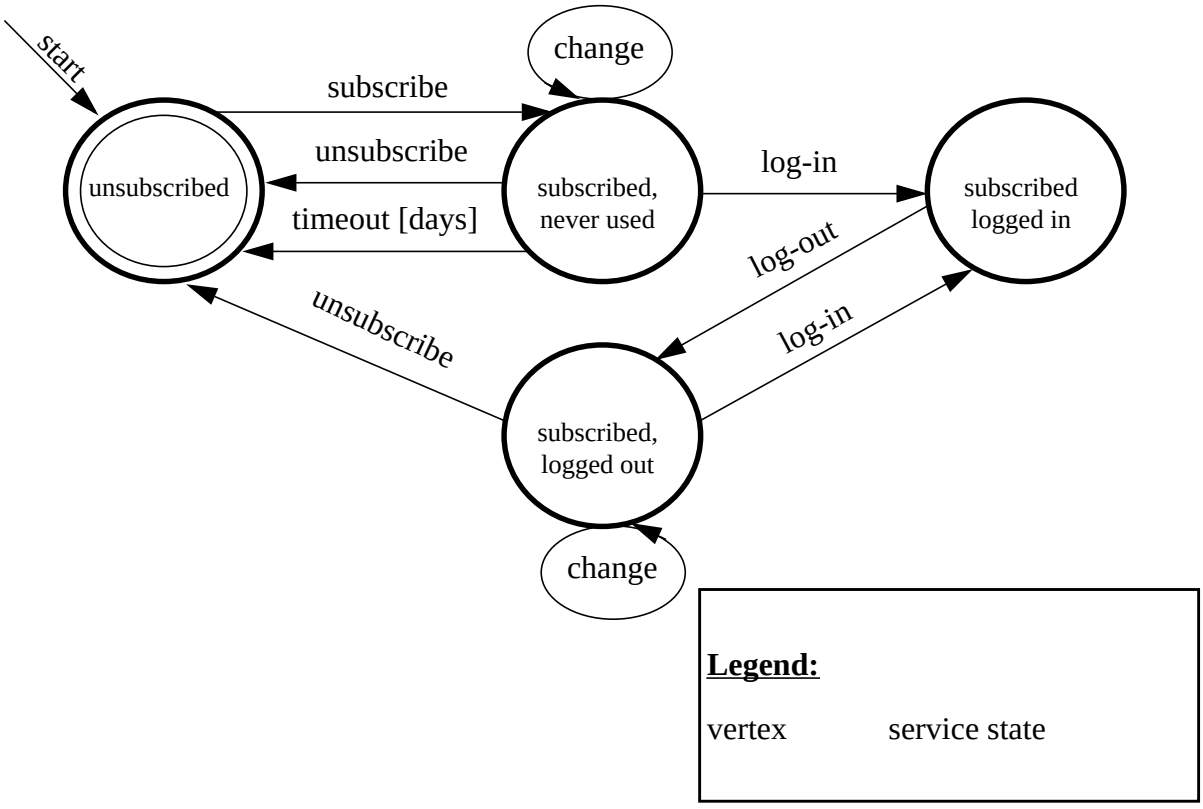

Figure 3: Service State Transition Diagram Extension

The correct usage of the service primitives is defined in the state diagram (Figure 2). Undefined transitions always cause a negative reply from the login-server (for example: subscribing to a service in the state “subscribed, logged out” causes the negative response “531 User already/ still subscribed”).

In future protocol versions, new service states might be defined. For example, the introduction of the state “*subscribed but never used*” might prevent charging a user for a never used service. This additional state is useful in the case where a user tries to subscribe to the service and the server accepts the subscription request, but the client considers the service unsubscribed due to an error (Figure 3).

2.2 Service Catalog

The ISMP protocol is designed for universal management of services provided to end-users over the Internet. The currently supported services are listed in Table 2, however the number of supported services can be extended either statically (i.e. with a protocol enhancement) or dynamically (see Section 3.5.2 on page 62 for more details).

The most important characteristics of each service are described in the Table 2:

- Service specific protocol. The Internet protocol required for accessing the service.
- Login actions. What is done when the service is logged-in in order to enable it locally.
- Logout actions. What is done when the service is logged-out in order to disable it locally.
- Subscription request parameters. The parameters which are requested by the server while subscribing to the service.
- Login response parameters. The parameters which are transmitted from the server to the NC while logging-in the service.
- Usage actions. The actions which are performed so that a user is able to use a service.

> *Note: All other service characteristics do not differ from each other in a currently defined service set. For example, no service-specific parameters are sent with a logout or unsubscription response for any of the services. Server actions are not addressed in this catalog either.*

Table 2: List of NC services supported by ISMP

service (ISMP service id)	service description	service specific protocol	login actions[a]	logout actions[b]	subscription request parameters	login response parameters	usage action
Provider service[c] (PROV)	In fact, PROV is no service. This "service" is used to identify a user to a provider. The user gets a user-id which is unique within the provider. Without subscribing to the PROV service, no other services may be subscribed to at the provider. Login and logout protocol operations are not defined for this pseudo-service.	ISMP	none	none	• last name • first name • address • telephone number • bank account • any number of any additional provider-specific information	none	none
Application Service[d] (APPL)	Service for renting applications.	HTTP, X-protocol	none	none	• application name	• applet URL	opening applet URL in a WWW browser (Netscape)

Table 2: List of NC services supported by ISMP

service (ISMP service id)	service description	service specific protocol	login actions[a]	logout actions[b]	subscription request parameters	login response parameters	usage action
Internet Connectivity Service[e] (ICS)	This service guarantees on-line access to Internet. All NC users must subscribe to ICS at a provider to be able to access the Internet and other NC services. All IP packets of unsubscribed users are filtered out. Additionally, OS update and an NFS directory for keeping persistent environment is provided to NC users.	IP	• updating OS (optionally) • setting local time according to net [f] • creating user local entry (passwd record) • mounting user's home directory and setting up NFS uid mapping [g]	• killing user's running processes • unmounting user's home directory • deleting user local entry (passwd record)	none	• user-id • exported home directory • exported OS directory • OS version • user's last name • user's first name	• creating invulnerable NC environment (chroot) • setting uid (su) • starting X environment (window manager, Netscape) Note: *this usage action is not started by the user but by the NC system after a successful ICS-login*
Mail Service[h] (MAIL)	Internet Mail Service	SMTP, POP	setting the default mail server in a browser (Netscape)	removing the mail server from the browser's preferences	none	none (in the future, the address of a POP server may be passed if it is different from the address of the SMTP server)	starting mail tool (Netscape message center)

Table 2: List of NC services supported by ISMP

service (ISMP service id)	service description	service specific protocol	login actions[a]	logout actions[b]	subscription request parameters	login response parameters	usage action
News Service[i] (NEWS)	Internet News Service	NNTP	setting the default news server in a browser (Netscape)	removing the news server from the browser's preferences	none	none	starting news tool (Netscape message center)
Disk Service[j] (DISK, WWW, FTP)	Service for renting disk place. The disk place can be optionally exported over a Web/FTP server to the Internet.	NFS, HTTP, FTP	mounting a remote directory and setting up NFS uid mapping	• killing applications accessing the remote disk[k] • unmounting the remote directory • resetting NFS mapper	• disk-size [kB] (number of blocks) • number of files (i-nodes) • backup-interval	• user-id • exported directory	starting file-manager (FileRunner)

a. Implemented in the method Service.afterLogin() which in turn calls shell-script /opt/FOKUSnc/sbin/<ISMP-service-id>-on with parameters returned by overridden method Service.getLoginPar().
b. Implemented in the method Service.beforeLogout() which in turn calls shell-script /opt/FOKUSnc/sbin/<ISMP-service-id>-off with parameters returned by overridden method Service.getLogoutPar(). Also script /opt/FOKUSnc/sbin<ISMP-service-id>-kill is called when necessary.
c. Implemented in class ProviderService
d. Implemented in class AppService.
e. Implemented in class ICSService.
f. rdate utility is used. The LS is queried for the time.
g. NFS mapper is implemented by slight modification of the mount utility and NFS kernel code.
h. Implemented in class MailService.
i. Implemented in class NewsService.
j. Implemented in class DiskLikeService and inherited classes: DiskService, WWWService, FTPService.
k. fuser utility is used.

2.3 The Transaction Model

To ensure the user's comfort and correct charging the ISMP must be reliable. The reliability is supported by the underlying protocol and *ISMP reliability means.*

The reliable transport-level protocol in the Internet protocol stack is TCP [rfc793] or its transaction-oriented extension T/TCP [rfc1379, rfc1644].

This chapter describes ISMP reliability mechanisms and examines 3 alternative usages of the underlying TCP protocol.

2.3.1 The ISMP Reliability Means

As is the case for every other protocol, ISMP should guarantee that both the client and server side negotiate an identical state. Unfortunately, no protocol can absolutely ensure this, and the danger exists that a server (or client) supposes that the other side is in a different state than it actually is. For someone using the ISMP this danger results in two possible *hazards*:

- *Usage hazard*. The server does not export a service, whereas the client claims that the service is available. This hazard causes different service-specific errors at the client side.
- *Charging hazard*. The client claims that a service is unavailable, whereas the server exports the service. This hazard can cause charging for unused services. (This hazard is considered more harmful than the former one — it causes high charges and is difficult to recognize.)

These hazards are minimized by the following *ISMP reliability means*:

- ISMP is a *3-way handshake (3WHS) protocol*[1]; every transaction consists of a request, response and acknowledgment. Acknowledgment is defined as the client's active close.

  Note: The ISMP acknowledgment is redundant — the correct arrival of an ISMP response is acknowledged at the TCP layer. However there is no mechanism to give a user feedback on incoming TCP acknowledgments at the socket layer. This compels a protocol designer to send the additional acknowledgment at the application layer. It does not have any effect on the total transaction time, because the client need not wait until the TCP acknowledgment of ISMP-ACK comes back. (The client can immediately send a FIN segment to finish the transaction.)

- The order of *local and remote operation* and transaction primitives is strictly defined to prevent the usage hazard. The server's login, subscribe and change operations (e.g. exporting a directory to an NC via NFS) must take place *before the* client's operations (e.g. mounting the exported directory). The server's logout and unsubscribe procedures must be executed *after the* client's procedures. (E.g. the server may unshare the exported directory only after the client unmounts it)

1. Implemented in ISMPoverTCP.makeTransaction().

- *Request and acknowledgment time-outs* are defined[2]. T*imeout-expiration actions* are defined.[3]:

Table 3: Timeout Expiration Handling

	login transaction	logout transaction
response timeout expires at the client side	• closing ISMP connection (the service is not set-up yet at the client side) • logout transaction is initiated to prevent usage hazard (if an error occurs it is ignored)	closing ISMP connection (the service is already reset at the client side)
acknowledgment timeout expires at the server side	resetting the service at server side (e.g. unsharing exported directory)	ignoring timeout (the service is already reset at the server side)

- In case of failure of the client's local login operations (e.g. if mounting an exported disk fails), or in the case of a negative ISMP reply, the same error handling is defined as for expired response timeout[4].
- The charging hazard is handled with "refresh cookies": The NC sends short "keep-alive" datagrams (the refresh cookies) for each logged in service (inclusive of ICS) to service servers at certain time intervals. Unrefreshed services are automatically logged out by the server. This turns the charging hazard into a usage hazard (A cookie may get lost or delayed, and the server may consequently log out the service.) which is considered to be less harmful.

  Note: Sending refresh cookies and automatic logging out will be disabled if the ISDN connection is temporally dropped in order to decrease the connection costs.

  The cookies also prevent another consistency problem: In the case of a serious failure (for example, a power failure or an X server failure) the NCSM might terminate without logging out all used services. If there were no automatic logout mechanism defined, the NC user would not be able to log-in the services again, because of the server's reply "service already logged in".[5]

The transaction model corresponding to the above defined constraints is illustrated in the enclosed transaction time lines (Figures 4 and 5).

All of the above defined reliability means together with the reliability features of the underlying protocol significantly minimize both hazards. The hazard situations (caused, for example, by in-

2. Implemented in ReliableProtocol.openConnection().
3. Implemented in Service.login and Service.logout().
4. Implemented in Service.login().
5. The cookies have been implemented as UDP packets in ISMP format (ISMP operation "COOKIE") on port 9012. These packets are sent by NCSM every 2 minutes for all logged-in services. Implemented in classes CookieTransaction and CookiesProtocol.

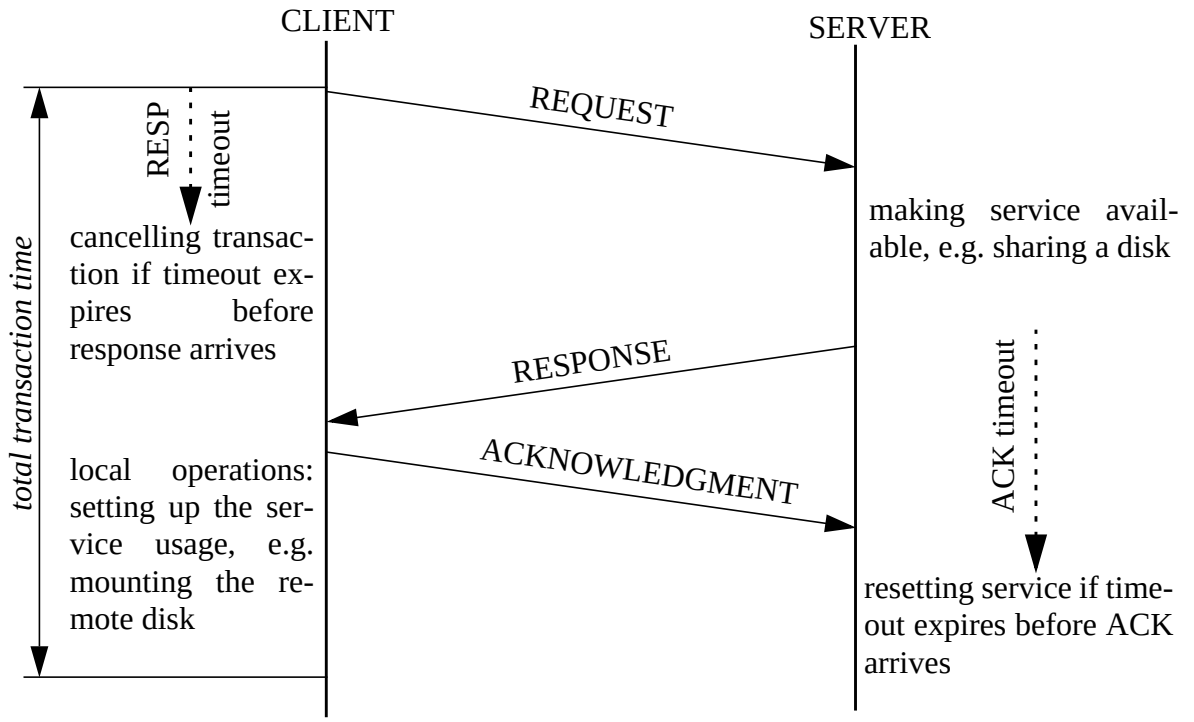

Figure 4: Time line of 3WHS protocol for login, subscribe and change transactions

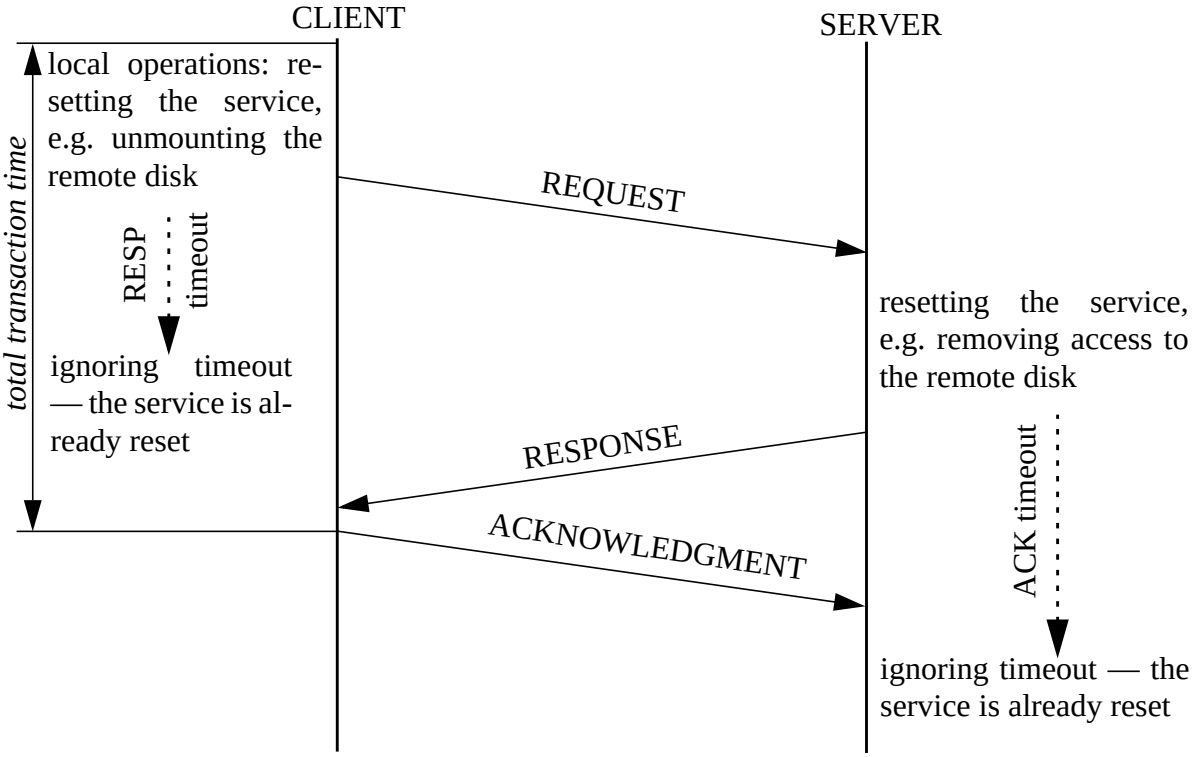

Figure 5: Time line of 3WHS protocol for logout and unsubscribe transactions

correct IP routing) are summarized in the enclosed Table 4.

Table 4: Summary of Transaction Hazards

Login Transaction	Logout Transaction
Situation: Request PDU lost or delayed, response timeout expires at the client side. Examples of reasons for failure: overloaded or unreachable LS, ISDN failure	
Server does not export the service, client does not use it, the transaction is cancelled by the client. => NO HAZARD	The client does not use the service any more, the server is not aware of that. The transaction is cancelled by the client. => CHARGING HAZARD Handling: No more cookies are sent to the server; after the server times out, the service is logged out.
Situation: Response PDU lost or delayed, response timeout expires at the client side. Examples of reasons for failure: incorrect routing from LS to NC	
The server already exports the service, the client cancels the transaction. But, the server does not recognize the erroneous state (according to ISMP definition in which ACK is defined as client's active close) and continues exporting the service. => CHARGING HAZARD Handling: The erroneous situation is recognized by the client and an automatic logout transaction is initiated.	Both the server and the client have reset the service, timeout expiration is ignored. => NO HAZARD
Situation: Acknowledgment PDU lost or delayed, acknowledgment timeout expires at the server side. Examples of reasons for failure: unreachable LS	
The client uses the service but the server resets it. => USAGE HAZARD Handling: The NC user is supposed to recognize the erroneous situation and manually log the service out.	Both the client and server have reset the service. The timeout expiration is ignored. => NO HAZARD

2.3.2 TCP Usage

There are three alternative methods for using the underlying protocol, TCP. They provide the same reliability but most importantly, they differ in the number of IP packets sent and the total transaction time.

All of the alternatives are illustrated at the IP level in the enclosed transaction time lines. CPT

stands for client processing time, SPT for server processing time, TPT for total processing time (TPT = CPT + SPT). All other abbreviations and names are common TCP and socket-API terms.

2.3.2.1 One ISMP Transaction over One TCP Connection (OoO)

Figure 6 on page 25 shows the time line for the most obvious alternative.

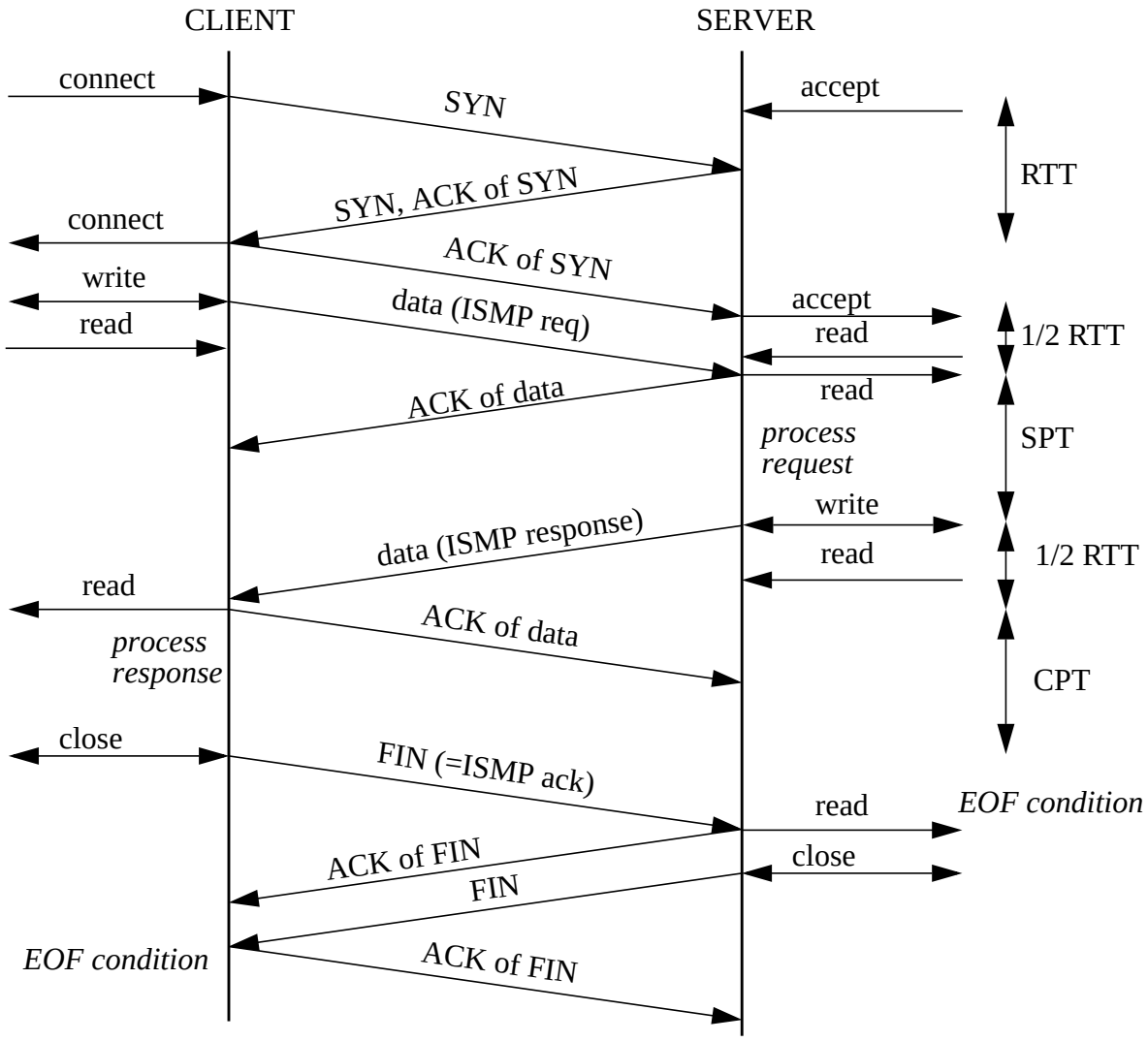

Figure 6: One ISMP Transaction over One TCP Connection

2.3.2.2 Many ISMP Transactions Over One TCP Connection (MoO)

Every TCP connection is initiated and finished using a three-way handshake. Having one TCP connection for every transaction produces overhead which can be reduced by sending many transactions over a single TCP connection. This might be especially useful during the *session*

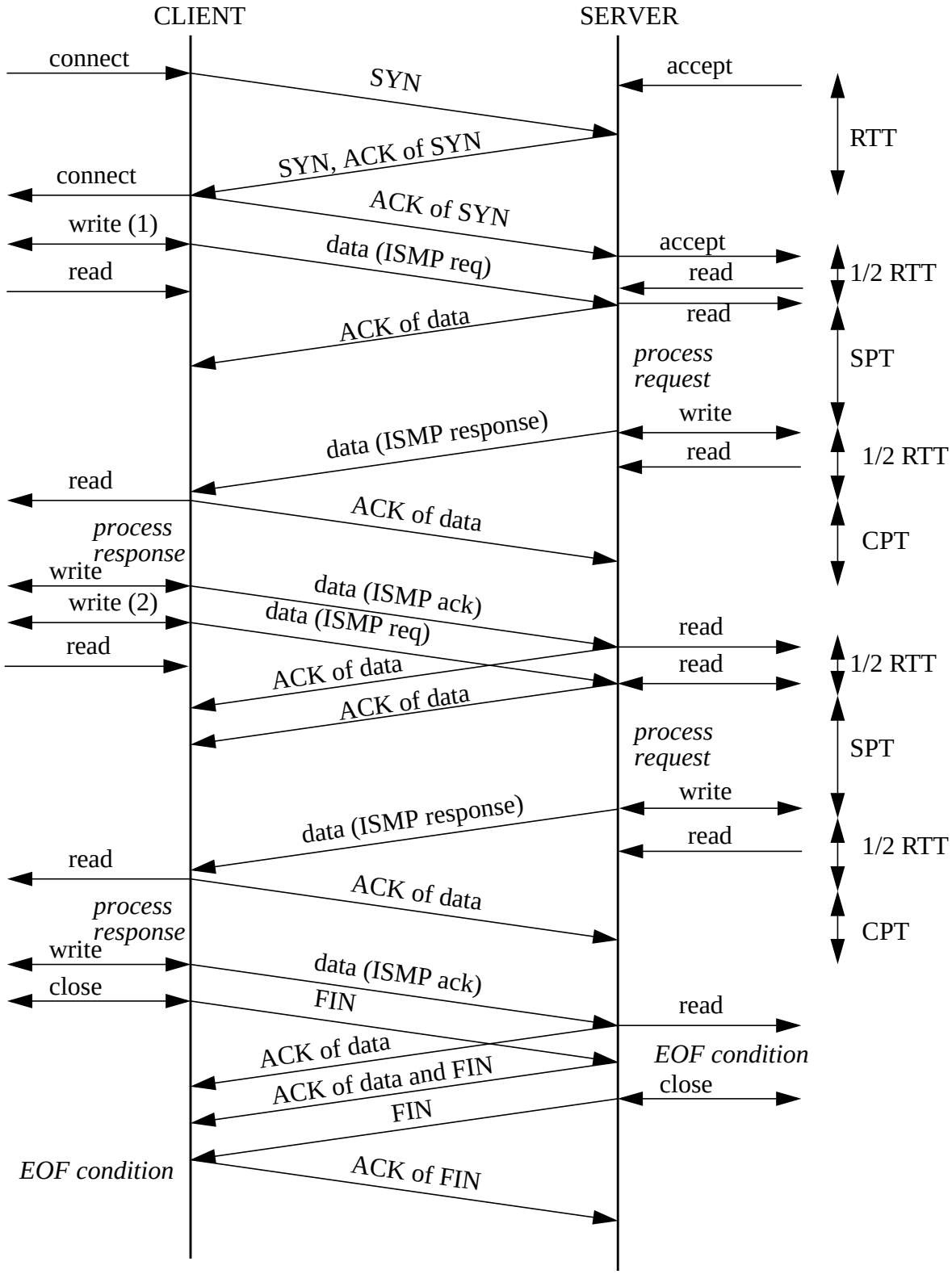

Figure 7: Many ISMP Transactions (2) over One TCP Connection

initiation phase in which the NC user logs many services in. The price for this alternative is more complex parsing. The time line of MoO is illustrated in Figure 7 on page 26.

> *Note: Interestingly enough, the overhead created by one client transaction over one TCP connection is only one RTT per transaction greater than that created by sending more requests over one connection. The reason is, that the system calls "write" and "close" on the client side do not block until a TCP acknowledgment arrives from the server. This means the transaction is already finished on the ISMP level, although the OS-kernel continues waiting for a finished TCP connection.*

2.3.2.3 The ISMP Transaction over T/TCP

"TCP for transactions, commonly called T/TCP, is an extension of TCP, designed to make client-server transactions faster, more efficient and reliable. This is done by omitting TCP's three-way handshake at the beginning of a connection and shortening the TIME_WAIT at the end of a connection." [SteIII, rfc1379, rfc1644]

Figure 8 on page 27 shows the time line for the "ISMP over T/TCP" transaction. One ISMP transaction is performed per T/TCP connection. T/TCP excludes ISMP acknowledgment.

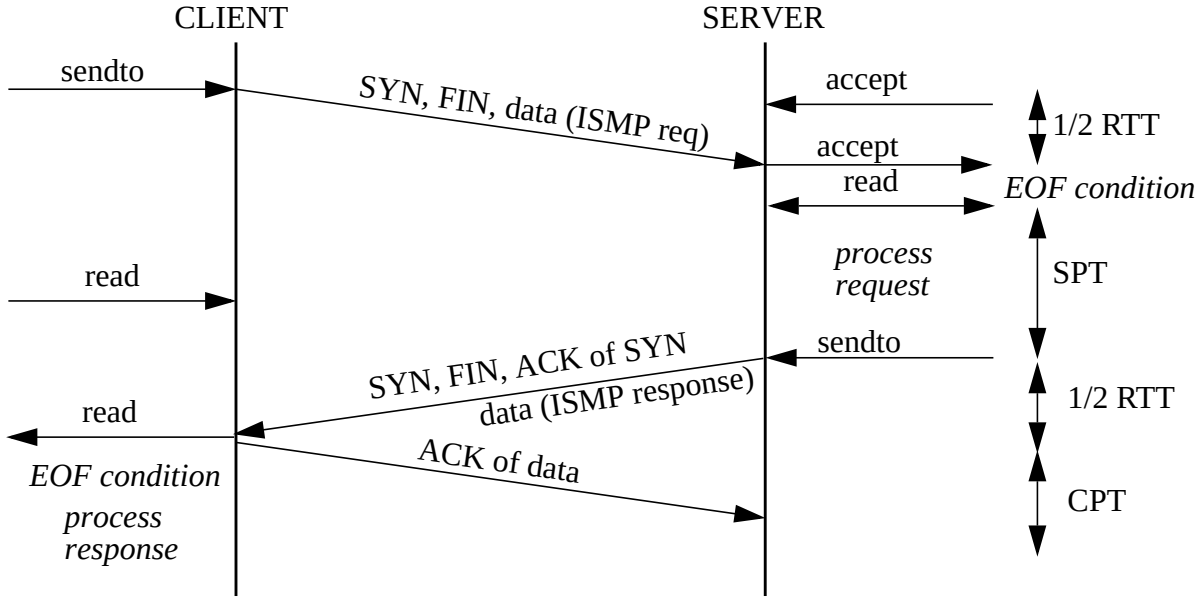

Figure 8: One Transaction over One T/TCP Connection

2.3.2.4 Evaluation

The three alternatives illustrated in the previous time lines are compared with respect to the total transaction time and number of exchanged IP packets. The transaction time is defined as the sum of the communication time (in terms of RTT) and both the client's and server's transaction processing time (Figures 4 and 5).

RTT over ISDN is considered to be approximately 30 - 40 ms (ping measured value). This value

is reasonable for services provided by a local provider — accessing services provided by other providers over the Internet may have longer RTT!

The initiation phase is considered to illustrate multiple transactions. It is supposed to consist of up to 10 transactions.

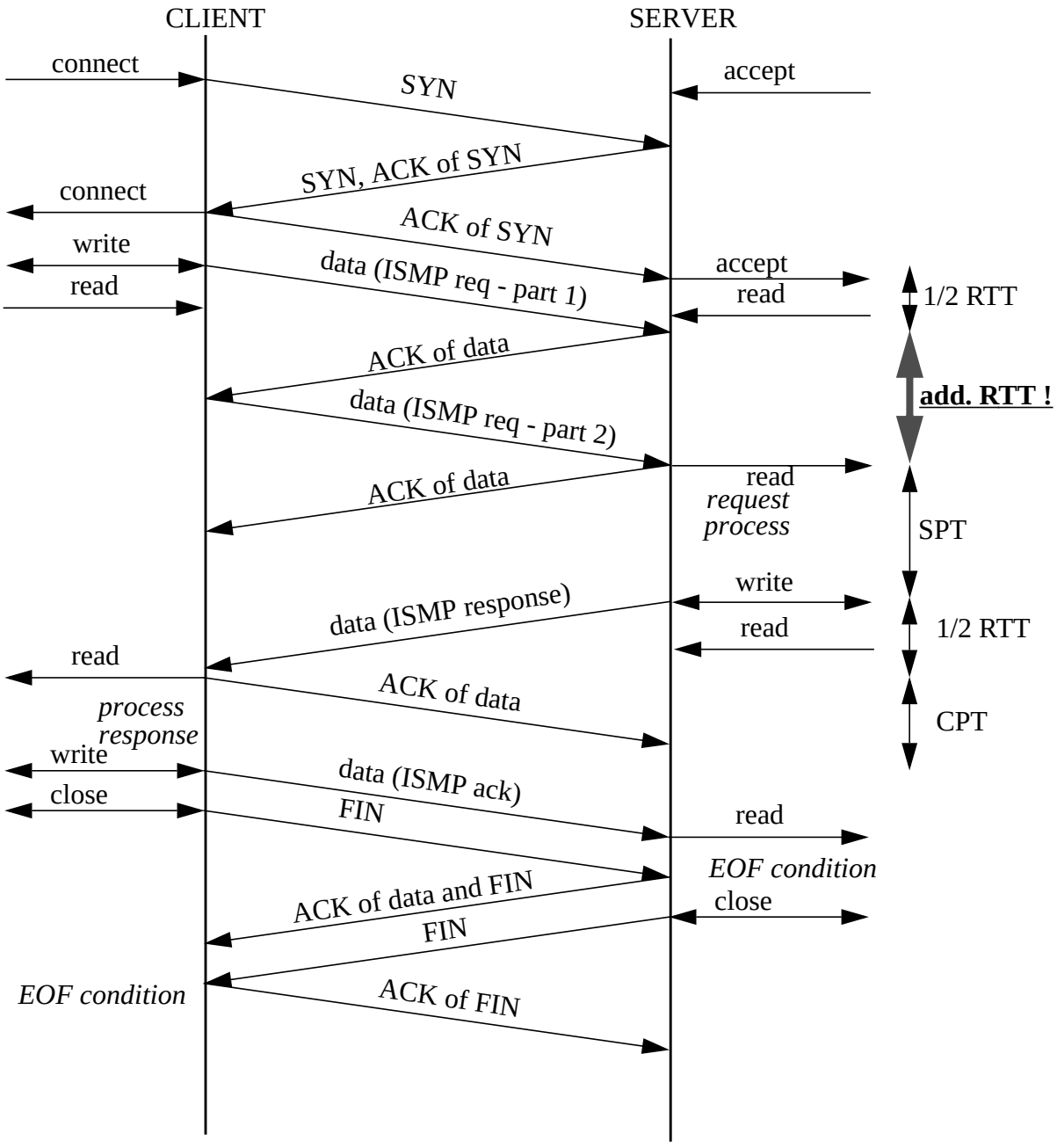

Figure 9: Sending a Large PDU

Note: The overhead calculation is simplified under the assumption that the PDUs are short enough to fit in MSS.

A PDU which is longer than the MSS will cause the TCP to work as poorly as a stop-and-wait protocol during the TCP's slow-start phase, because the TCP window size

is set to one MSS. This increases the transaction time in terms of RTT (the additional RTT delay is pointed out in the enclosed transaction time line).

However, sending more short TCP segments (for example sending a PDU field after field) does not introduce this delay until the total length becomes larger than the MSS. The only affected quantity is the number of transmitted IP packets.

See the time line (Figure 9 on page 28) for more details.

Table 5: Total Transaction Time

	total time of N transactions [ms]	total time values for 10 transactions with RTT = 40 ms [ms]
one transaction over one TCP connection	N*(2*RTT+TPT)	10*(2*40+TPT) = 10*TPT + 800
N transactions over 1 TCP connection (see the next note on page 29)	RTT+N*(RTT+TPT)	40 + 10*(40+TPT)= 10*TPT + 440
one transaction over one T/TCP connection	N*(RTT + TPT)	10*(40 + TPT)= 10*TPT+400

The absolute winner with the minimum transaction time is definitely T/TCP. However, the profit is marginal [tenths of seconds] in comparison to the other alternatives and can therefore be neglected under the assumption of an ISDN RTT of 40 ms.

Note: When running several transactions over separate TCP connections (e.g. at the beginning or end of a session) the total time can be significantly reduced by parallel transaction processing (This is what Netscape does when it sends multiple HTTP requests at once.).

Table 6: Number of IP Packets of ISMP Transaction

	number of sent IP packets	number of IP packets sent during 10 transactions
one transaction over one TCP connection	11*N	11*10 = 110
N transactions over 1 TCP connection	7+6*N	7+6*10 = 67
one transaction over one T/TCP connection	3*N	3*10 = 30

As for the number of exchanged IP packets T/TCP is the winner again. However, this aspect does not matter when using a dedicated ISDN channel.

Conclusion:

T/TCP is definitely the most efficient of the three alternatives. However, the real profit is marginal when using a dedicated 64 kbps ISDN channel, and any other alternative can be used without a notable reduction of the total protocol efficiency. Because implementations of T/TCP are rare (no support in JDK) and the “N over 1” alternative is more difficult to implement, the “1 over 1” alternative has been selected for ISMP, although its efficiency is the poorest. When necessary, the efficiency can be drastically improved with parallel transaction processing.

2.4 Protocol Format

There are basically two format types: *textual* and *binary*. The binary format is faster (no parsing required, binary representation is shorter), the textual format is *portable* and *easy to debug and administer*. The advantages of using the former representation outweigh those of using the latter representation only in the case of transmitting large amounts of data, which is definitely not the case with ISMP. Therefore ISMP is a *textual* protocol similar to protocols like SMTP or HTTP.

There are three approaches to building and parsing PDUs:

- fixed-length PDUs (Note: Fast processing but low format flexibility and extensibility. Suitable especially for transport protocols but *not* for high-layer protocols.)
- variable-length PDUs with a field “size of PDU” (Note: There are problems with encoding the “size” field.)
- variable-length PDUs with a delimiter on the end

The “*variable-length PDU with a delimiter*” alternative has been selected — it does not suffer from any of the disadvantages of the two other alternatives. EOL has been chosen as the PDU delimiter. The same character delimits each PDU field as well (this implies two EOLs at the end of every PDU). EOL is defined as “line-feed” character. For maximum portability the pair “carriage-return”-“line-feed” shall be also recognized by PDU receivers.

Such a format causes the following steps to be performed while receiving a PDU:

- reading line after line until a line is empty (in C: with *fgets*; in Java: with DataInputStream.readLine)[6]
- deleting EOL from each line, throwing away leading spaces and tabs
- decoding PDU[7]

Each PDU consists of the following four field-groups[8] (Figure 10; some of them may be empty in a PDU) which are tagged with different patterns to distinguish them from other items in a detailed protocol definition.

6. Implemented in Transaction.readResponse().
7. Implemented in Transaction.parseResponse().

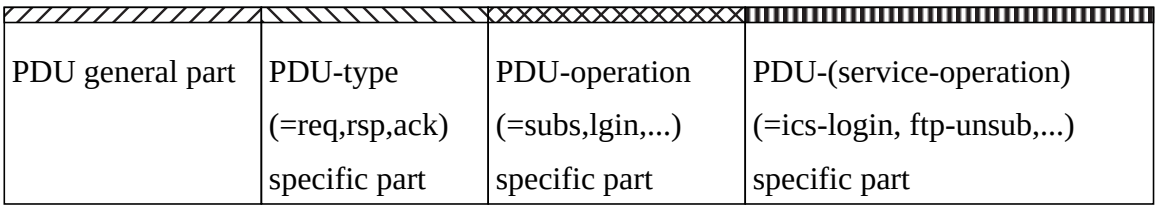

Figure 10: ISMP Format

2.4.1 ISMP Request Format

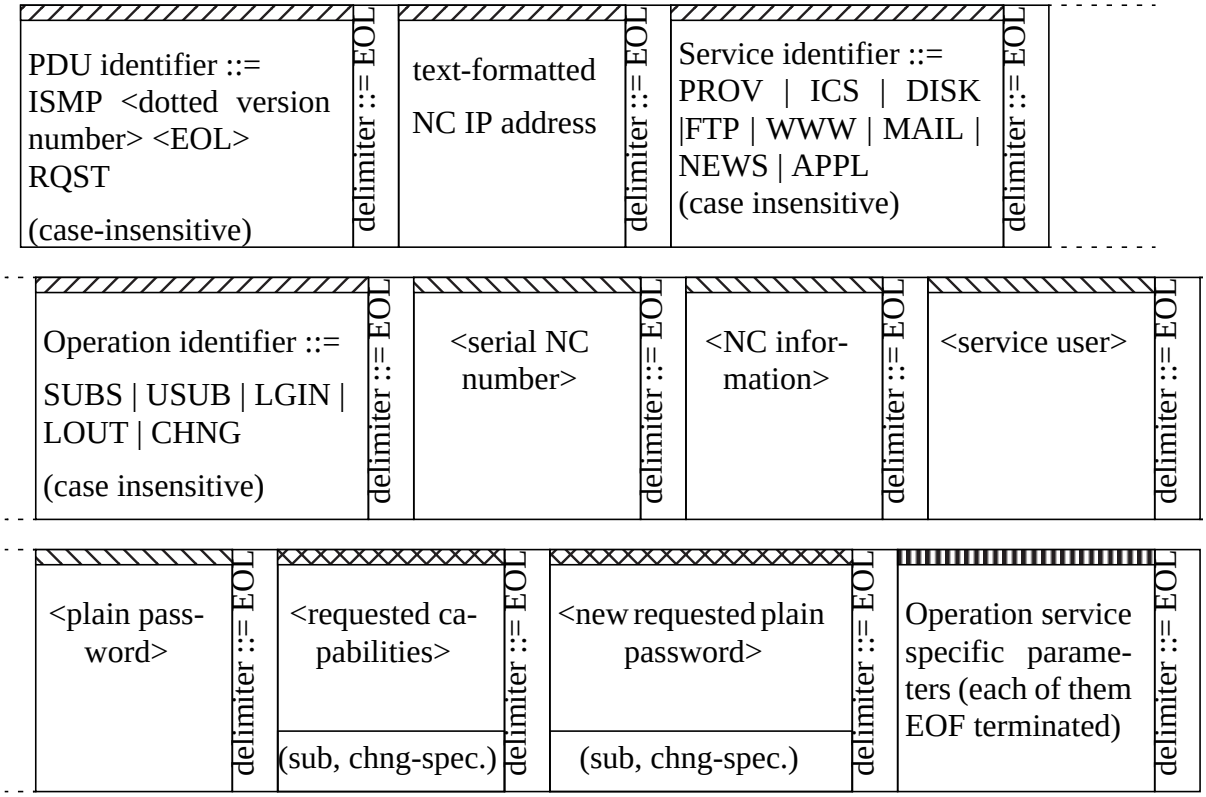

Figure 11: ISMP Request Format

8. Both *general* and *type-specific* PDU parts are implemented in Transaction.printRequest() and Transaction.parseResponse().

The *operation-specific* part is implemented in methods overriding Transaction.printOperationSpecificPars() and Transaction.parseOperationSpecificPars().

The service-operation-specific part is implemented in methods Transaction.printServiceSpecificPars() and Transaction.parseServiceSpecificPars() which in turn call methods overriding Service.<print|parse><Sub|Usub|Login|Logout|Change>SpecificInfo() and optionally methods overriding Service.ServiceParameters.<parse|print>().

Notes:

- The <*NC IP address*> field is defined only within the provider's area. Its value must be ignored by LS while receiving the PDU from NC, the source IP address delivered in the IP packet will be used instead. This prevents trouble in cases where the NC can not be directly reached under its IP address, for example when IP masquerading (refer to Section 3.1.6 on page 46 for additional details) is enabled. (But: the field must not be empty — the end of the PDU would be recognized by the LS.) Malicious users are prevented from fooling the LS at the ISMP level as well.
- The <*serial NC number*> field identifies the NC uniquely. This can be used as an efficient mean for robbery protection for example. ICS providers would keep track of all stolen NCs (identified by their hard-wired NC number) and would immediately alarm if such an NC connected.
- The <*NC information*> field provides the server with additional information on the NC's hardware and software. Currently, the format of this field is defined as the output of the Unix command "uname -srm".
- The <*service user*> field contains a unique username within a provider. In the provider-subscription PDU it is a proposal for a new username. (This request might be rejected if there is already another subscriber with the same username). In all other PDUs the field refers to the which was username proposed by the user and confirmed by the provider.
- The <*password*> *field* includes the *provider-specific password* for the operations SUBS, USUB and CHNG or the *service-specific password* for the operations LGIN and LOUT. Distinguishing between provider-specific and user-specific passwords enables two-level security. A more privileged person can perform all operations. (She or he must know the provider-password for the operations SUB, USUB and CHNG by heart.) A less privileged person can only log-in and log-out the services and is not allowed to perform the other operations. (She or he does not need to remember any password — the service-specific passwords are stored in a configuration file.)
- The "*requested service capabilities*" field specifies additional parameters, which will be transmitted by the LS using the response-specific "*capabilities*" field. This is a catchall for all values not defined in the ISMP, e.g. those for extended provider-specific charging information.
- The <*new requested password*> *field* is used for the SUBS and CHNG operations only. In the special case of provider subscription (the NC user is not assigned any password yet), this field will have the same value as the <*password*> *field.* These two fields also have identical values when a CHNG operation is requested in order to change service's parameters, and the service-specific password remains unchanged.

2.4.2 The ISMP Response Format

Notes:

- *Status codes* are defined according to Internet convention (SMTP, HTTP, ...) as a 3-digit integer result code *xyz*. The first digit (x) of the status-code defines the class of response (1-positive preliminary, 2-positive completion, 3-positive intermediate, 4-transient negative completion, 5-permanent negative completion). The second digit (y) defines the class of reason for the error code (0-syntax, 1-Information, 2-Connections, 3-Authentication and accounting, 4-Unspecified yet, 5-Local System). The last digit (z) specifies the reason for the

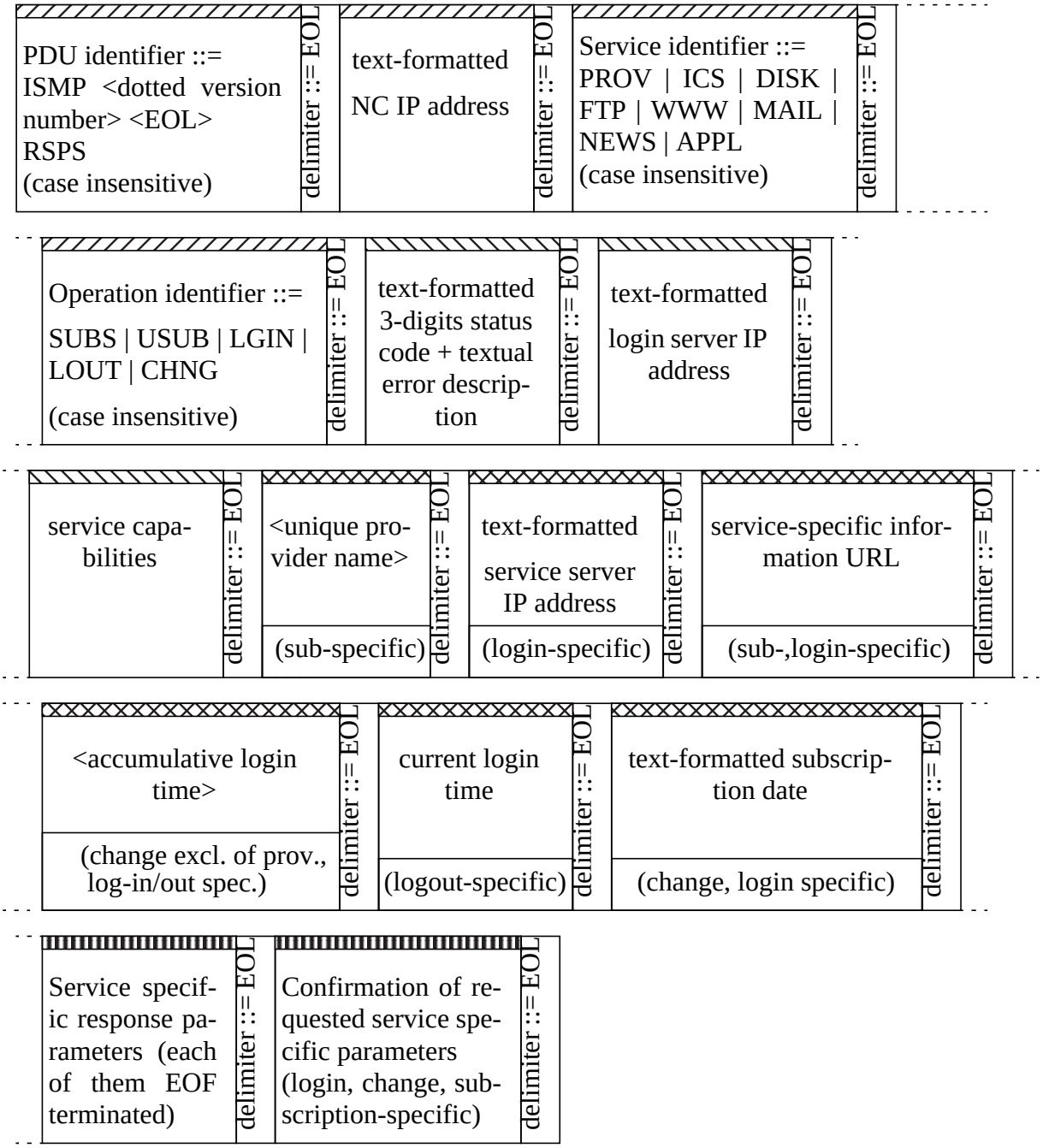

Figure 12: ISMP Response Format

error code. A complete list of error code definitions is contained in Table 7.

Table 7: ISMP Response Codes

Status Code	Meaning
200	Success
420	Socket error

Table 7: ISMP Response Codes

Status Code	Meaning
421	Connect error
451	fdopen error
500	Invalid ISMP PDU
501	Client ISMP error
502	Server ISMP error
530	User not subscribed
531	User already/still subscribed
532	Invalid user/password
533	Application does not exist
550	Passwd file does not exist
551	Passwd file access error
552	Create directory error
553	Share directory error
554	Service login database error
555	Set quota or backup interval error
555	Add user to htpasswd file error
556	Get/put uid error

- *Service capabilities* are a field used to transmit special values not defined in the ISMP (e.g. provider-specific detailed charging information). These values are requested in the ISMP request's capability field.
- The *service specific information URL* is intended to provide NC users with a source of the most recent information on the subscribed service. The NC receives this field while subscribing for the first time. Its initial value is regularly updated at every login.
- The *service server address* field guarantees dynamic assignment of service-specific login servers. The NC never knows which server it will be assigned to — this information comes only from the protocol. For example, an NFS server can be replaced by a more powerful one, if the user's disk usage exceeds its capacity without any noticeable effect for a user.
- The "*confirmation of requested service specific parameters*" consists of the confirmation of all service-specific parameters requested from the server in the "service-specific parameters" field group of the subscription-request PDU. This is used for the detection of possible inconsistencies between required and actually available service parameters. (It causes the following parameter groups to be identical: subscription-request service-specific-parameters, change-request service-specific parameters, login-response confirmation parameters, change-response confirmation parameters and subscription-response confirmation parame-

ters.)

There is one exception to this rule: no parameters are confirmed at provider-subscription, in order to avoid sending privacy-sensitive data over the network.

- "*Accumulative Login Time*", "*Current Login Time*" and "*Subscription Date*" are values sent by the LS to allow the NC user to control service usage.

2.4.3 A Transaction Example

Figure 13 is an example of an ICS logout transaction for user "adam".

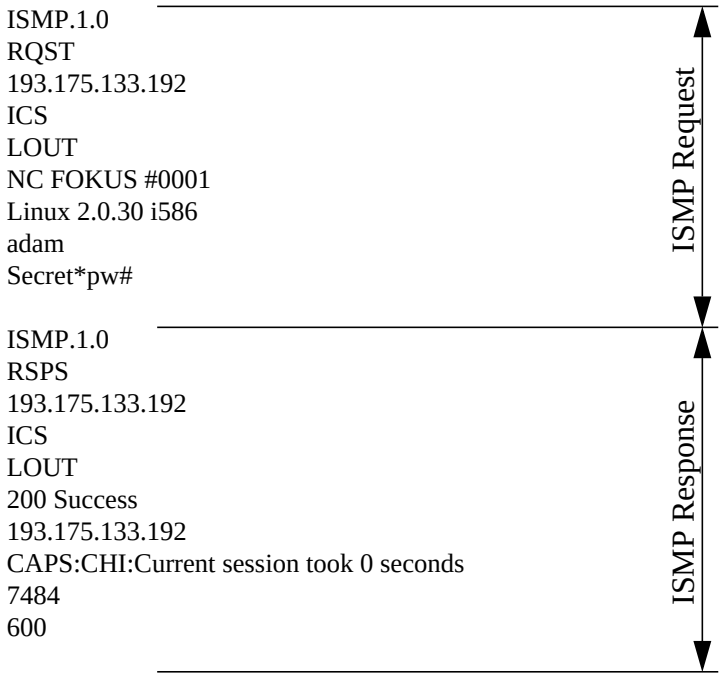

Figure 13: ISMP Transaction Example

3 Technical Issues of NC Client

The previous chapters discussed the general concepts of internetwide network computing. This chapter discusses various technical issues of the concept in detail. The solutions to these implementation-independent issues are a part of the concept.

3.1 Internet Connectivity

Internet connectivity is the basic requirement for accessing all other services offered over the Internet. IP traffic must be routed, the name-server must map host-names to their IP addresses, one's personal environment must be consistently available at any Internet access point, and so on. These points are discussed in the subsequent subchapters.

3.1.1 Network Architecture

The network provider is responsible for the connection of the NC to the network. Therefore, he has to guarantee at least all of the services in Table 8:

Table 8: Required Services Provided by ICS Provider

Service	Means to Afford the Service
ISDN and PPP connectivity: This service is necessary to provide an NC with a physical connection to the on-line provider.	ISDN router DHCP server
ISMP Login Service: This service is the basic mechanism for renting NC-services over Internet.	Login Server
Name service: This service is required in order to be able to map hostnames to their IP addresses and vice versa. In particular the IP addresses of the LS and WWW server must be mapped from their names "logins" and "www". (See Section 3.1.3.3 on page 42 for more details.)	DNS server
WWW: This service is required in order to give an NC-user a chance to identify himself and dynamically subscribe to ICS and other services offered on Web-pages with embedded Java applets. (See Section 3.3 on page 57 for more details.)	HTTP Server
IP forwarding: This service guarantees Internet access to subscribed users. (Packets of unsubscribed users are filtered out.)	IP router with enabled filtering

Except for the IP forwarding, all these services are freely available to any users who dial up a provider's server over ISDN lines. This enables new users to use the WWW server to subscribe at the provider.

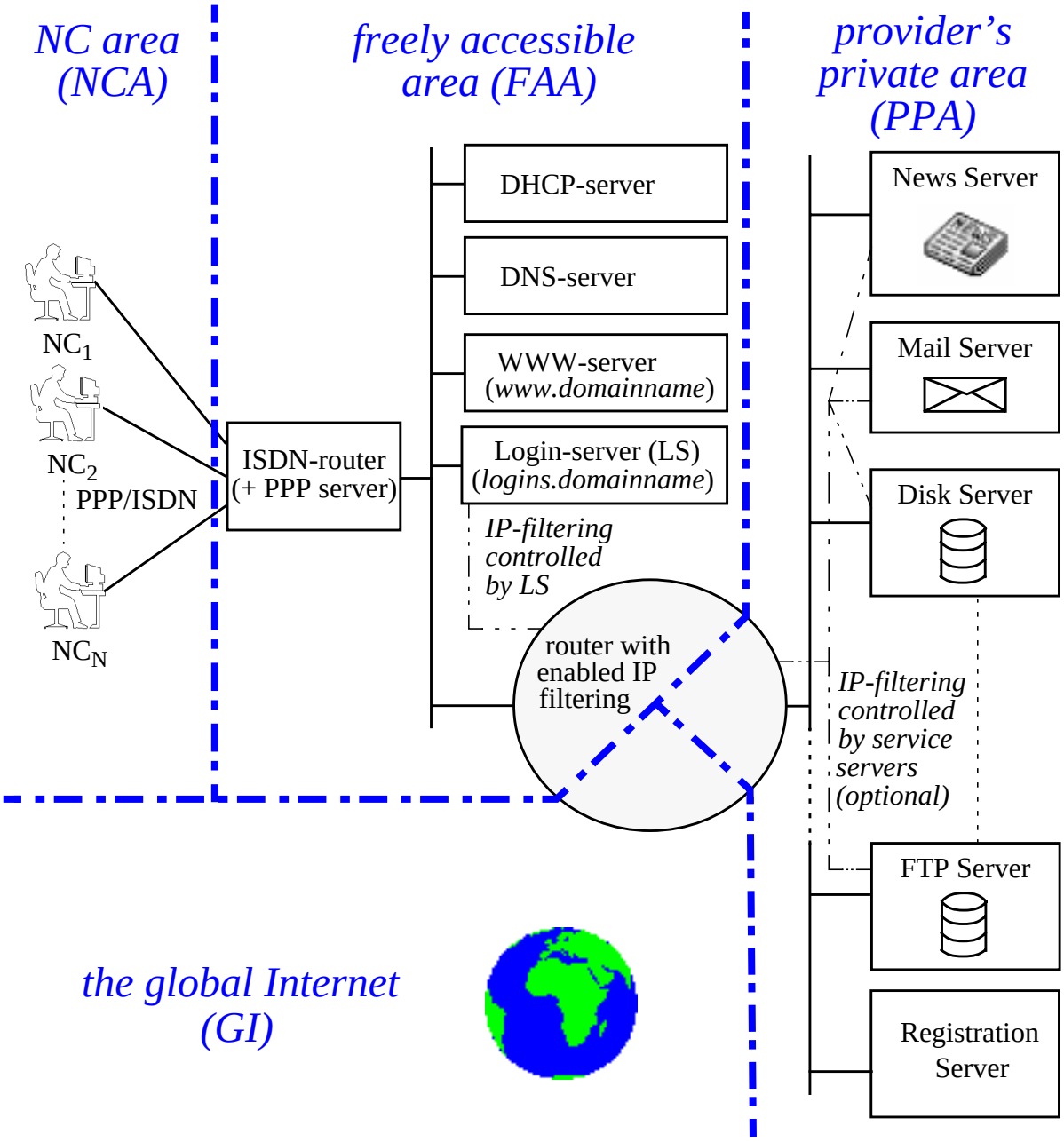

Figure 14: Network Architecture

A design of network architecture meeting these requirements is depicted in Figure 14. It consists of four areas: *NC area (NCA)*, *freely accessible area* (FAA) for subscription purposes, protected *provider's private area* (PPA) for service-specific servers and *global Internet* (GI). These areas are physically separated and IP traffic between them is run over a router with enabled packet-filtering controlled by the LS (see subsequent Section 3.1.2. for details on IP-filtering).

Note: A host may be shared by several servers; DNS and DHCP servers may be run using a single host, for example.

Note: The ISDN router has to be capable of filtering out all IP-packets with a source-address differing from that assigned by the PPP daemon. Otherwise, malicious users might send out IP packets with the IP address of other currently logged-in users.

IP packets of users who have not subscribed to *Internet Connectivity Service* (ICS) are not allowed to leave the FAA. New NC users may subscribe to the ICS over the freely accessible WWW server. After logging-in to ICS, their packets will be forwarded to both of the other areas behind the IP router: PPA and GI. The benefit of this solution is that *purely electronic* means (publicly accessible PPP, DNS, LS, DHCP and WWW server) are needed for subscribing at a provider (see Section 3.3 on page 57 for more details on subscription).

3.1.2 IP Filtering

There are two reasons for enabling IP-filtering at the provider's router:

- to control the Internet access of ICS subscribers (i.e. only packets of users who have subscribed to and logged-in to ICS may be forwarded);
- to protect the provider's infrastructure against security attacks.

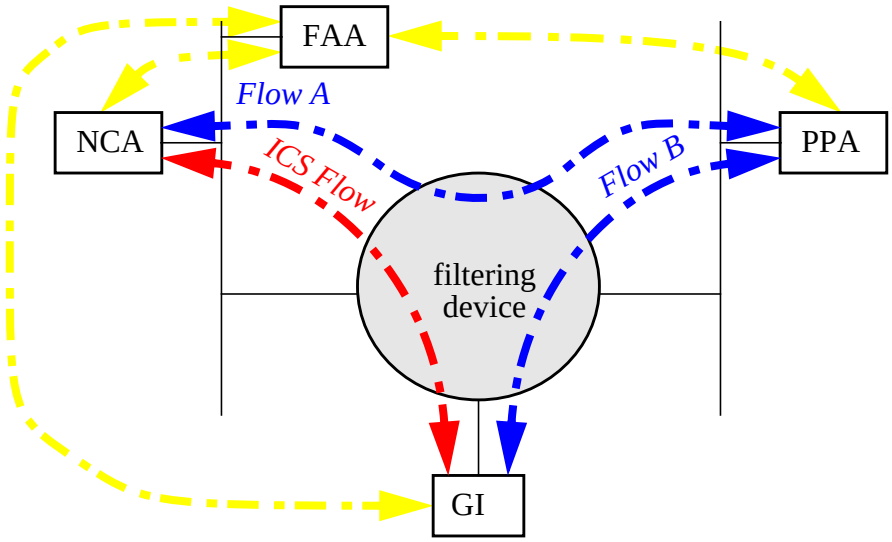

Figure 15: Packet Flows

This subchapter identifies packet flows (Figure 15) between areas defined in the previous chapter and suggests filtering strategies to be applied to them. The filtering rules are to meet the following requirements:

1. *The FAA is always open to the NCA and GI in order to support automatic subscription. Also communication between the LS and the PPA is free (yellow flows in Figure 15).*
2. *The packet-flow between the NCA and GI (red ICS flow in Figure 15) is always controlled by the LS in order to allow Internet access to ICS users only.*

3. *The PPA packet flows may be optionally controlled by service servers; identical rules are to be applied to both flows (blue flows A and B in Figure 15). The reason is that it does not make sense to fortify only one of these flows with IP-filtering, if an intruder may attack the other unprotected one.*
4. *Filtering rules should be as simple as possible in order to minimize filtering overhead, resulting in higher packet-delay.*

The requirements imply all the rules except for the PPA rules summarized in Table 9. The two rules controlling access to PPA (packet-flows A and B) are discussed in the following paragraphs.

Table 9: IP Filtering Rules

area	NCA	PPA	GI
FAA	free (for subscription purposes)		LS free
GI	only packets of users who have logged-in ICS are forwarded; IP-filtering is controlled by LS (ICS Flow)	provider defined flow B (IP filtering controlled by service servers)	
PPA	provider defined Flow A (IP filtering controlled by service servers)		

In general, it is up to the provider to choose an individual strategy for flows A and B, depending on the power of his filtering device. A more restrictive filtering policy (i.e. one containing more complex filtering rules) guarantees a higher degree of security, but introduces higher filtering overhead and results in higher packet delay. Possible strategies (ordered from the most to the least restrictive) are suggested in Table 10. As you can see, even the simplest filtering strategy *iv* requires as many rules as the number of hosts connected to provider, if *dynamic filtering* is enabled. This may introduce a significant overhead for most of today's routers, if hundreds or even thousands of users are connected.

> *Note: The delay dependence is very difficult to quantify, because it depends on too many factors. The most important factors are the router's architecture, the number of filtering rules, filtering depth and the current router's load.*

The answer to the problem is to detach the NCA in the *verified* and *unverified subnet* (*VSN, USN*), filter out packets of the unverified subnet, and assign VSN addresses only to NCs which authenticate themselves (the authentication can take place at the PPP level).

Such a solution provides the same security as the strategy iv, but requires only one IP-filtering rule![1] The low overhead makes this alternative the most reasonable.

1. Additional implementation overhead (PPP authenticating and address assignment controlled by the LS) has to be taken in account.

Table 10: Filtering Strategies for PPA (Flows A and B in Figure 15)

Strategy	Filtering Rules for a Packet-Flow (allowed IP-packets)		Comment	total number of filtering rules (H_X is the set of hosts accessing the PPA over flow X; s_i is the number of PPA servers accessed by client i)
	A	B		
i	∀ {NCA host, PPA host & port}	∀ {GI host, PPA host & port}	The most secure policy with the highest filtering overhead (UDP/TCP headers have to be examined, an entry exists for each PPA host.).	$\|H_{ICS}\| + \sum_{\forall(i \in H_A)} s_i + \sum_{\forall(i \in H_B)} s_i$ (incl. port numbers!)
ii	∀ {NCA host, PPA host}	∀ {GI host, PPA host}	Filtering overhead and security degree is lower than in the strategy i, because only the IP header is examined.	$\|H_{ICS}\| + \sum_{\forall(i \in H_A)} s_i + \sum_{\forall(i \in H_B)} s_i$
iii	∀ {NCA host, PPA subnet}	∀ {GI host, PPA subnet}	Here, NCA/GI hosts are allowed to access the entire PPA after having identified themselves by logging-in a service. The A-rule {NCA-host, PPA subnet} can be combined with the ICS-rule {NCA-host, GI} resulting in a simple efficient rule {NCA-host, *}, under the assumption that NCA hosts are not allowed to access PPA without first logging-in to ICS.	$\|H_{ICS} \cup H_A\| + \|H_B\|$ or $\|H_{ICS}\| + \|H_B\|$ (if A-rules are combined with ICS rules)
iv	free	free	Minimum security degree; other security means have to be used unconditionally (e.g. TCP Wrapper [tcpw]). Because the PPA is freely accessible, it can be integrated in one segment with FAA.	$\|H_{ICS}\|$ (ICS-rules only)

3.1.3 Session Initiation

There are a couple of points related to the initiation of an NC-session.

3.1.3.1 IP Address Assignment

Before accessing the Internet an IP address must be assigned to an NC. An address from a PPP pool can be assigned dynamically over the PPP IP Control Protocol [rfc1332]. Unfortunately, there is a pitfall: If the ISDN connection is interrupted (due to a failure or for saving reasons during idle periods), a completely different IP address is assigned to the NC after the connection is reestablished. This causes all established TCP connections to fail.

This problem can be solved on the server side by caching the NC's telephone number and reserving an IP address exclusively for this telephone number for a certain period. This may be done by a DHCP server, which is always queried by the PPP server upon connection establishment. A detailed solution will be available in the next project stages.

> *Note: This concept supports mobility — NC users can access their services from any Internet access point offered by their provider. (In the future, they may use the IAP of any provider, see Section 3.1.4 on page 43.) This has nothing to do with the so called Mobile IP [rfc2002]. NCs get their IP addresses dynamically assigned to them whereas mobile nodes keep their home addresses. NC users can use any available NCs; mobile node users have to travel with their devices.*

3.1.3.2 Adding Default Route

When IPCP [rfc1332] negotiation is completed successfully, NC's PPP daemon (pppd) will inform the kernel of the local and remote IP addresses for the PPP interface and set up the peer's IP address as the default NC's route[2]. The route is deleted upon link termination by pppd as well.

3.1.3.3 Setting-up Name Resolver

To access network hosts by name, a name-server must be available and the NC's name-resolver must be set up, i.e. the IP address of the name-server and a domain name must be known to the NC's name-resolver.

The IP address of the DNS server can be obtained over extended [RFC1877] version of PPP's [RFC1661] IP Control Protocol [RFC1332]. (Support at the provider's ISDN router is required!) The PPP server itself may receive the address queried by the client over a boot-protocol, either the extended [RFC1497] version of BOOTP [RFC951] or its ancestor DHCP [RFC1531].

2. The "defaultroute" option of pppd has to be set.

After having received all the necessary information, the NC's name-resolver can be successfully set up[3] and the NC can query its hostname (and consequently the domain name) at the name-server.

3.1.4 Service Database

An NC has to keep track of all of a user's subscribed services in order to be able to control them. Data transmitted from the LS over the ISMP (e.g. service-specific parameters) have to be cached until they are updated. This information must be available at anytime at any Internet access point (IAP). Therefore, a "service database" (SDB) is required. Currently, a SDB is defined to be an NFS distributed, text-formatted database. The NFS directory in which the SDB is stored[4], is provided along with the ICS as the user's *home directory*. All other data which ensure that the user's environment is consistently the same (e.g. Netscape preferences) are also stored in this directory.

The format of the SDB is defined in the syntax diagram in Figure 16[5]. Some of the trivial non-terminal fields, e.g. the subscription date, are not defined in detail. Additionally, any line beginning with the hash character ("#") is completely ignored according to Unix convention.

As you see, the current SDB design is quiet simple. The simplicity has several drawbacks which should be corrected in upcoming project stages. The drawbacks are:

- SDB access is *not synchronized*, i.e. the SDB may become inconsistent if it is simultaneously accessed by several NCs. In other words, the unsynchronized SDB works only under the assumption that only one user is allowed to log in to the ICS at one time. In the future, record locking or a completely different database system may be used.
- *Integration of the SDB with the ICS* allows an NC user to have only one ICS provider — users wanting to access the Internet over the IAP of another provider (for instance travelling salesmen) cannot access their SDBs. This could be fixed by separating SDB from the ICS and providing it as a special service. The separation would also allow users who change their ICS provider to keep their SDBs. Currently, they have to unsubscribe all other services before unsubscribing ICS. Additionally, a separate SDB would make it possible to access the PPA without first logging in to the ICS.

All of these disadvantages indicate that the SDB should be redesigned soon. A straightforward solution addressing all of the problems discussed above would be to store the SDB on a personal *chip-card* along with the user's encryption keys.

---

3. In our implementation, the NC's PPP daemon is patched to modify the file "/etc/resolv.conf".
4. The SDB is saved as "~/.nc/services.nc".
5. A new SDB format is currently being discussed; it differs only slightly from Figure 16.

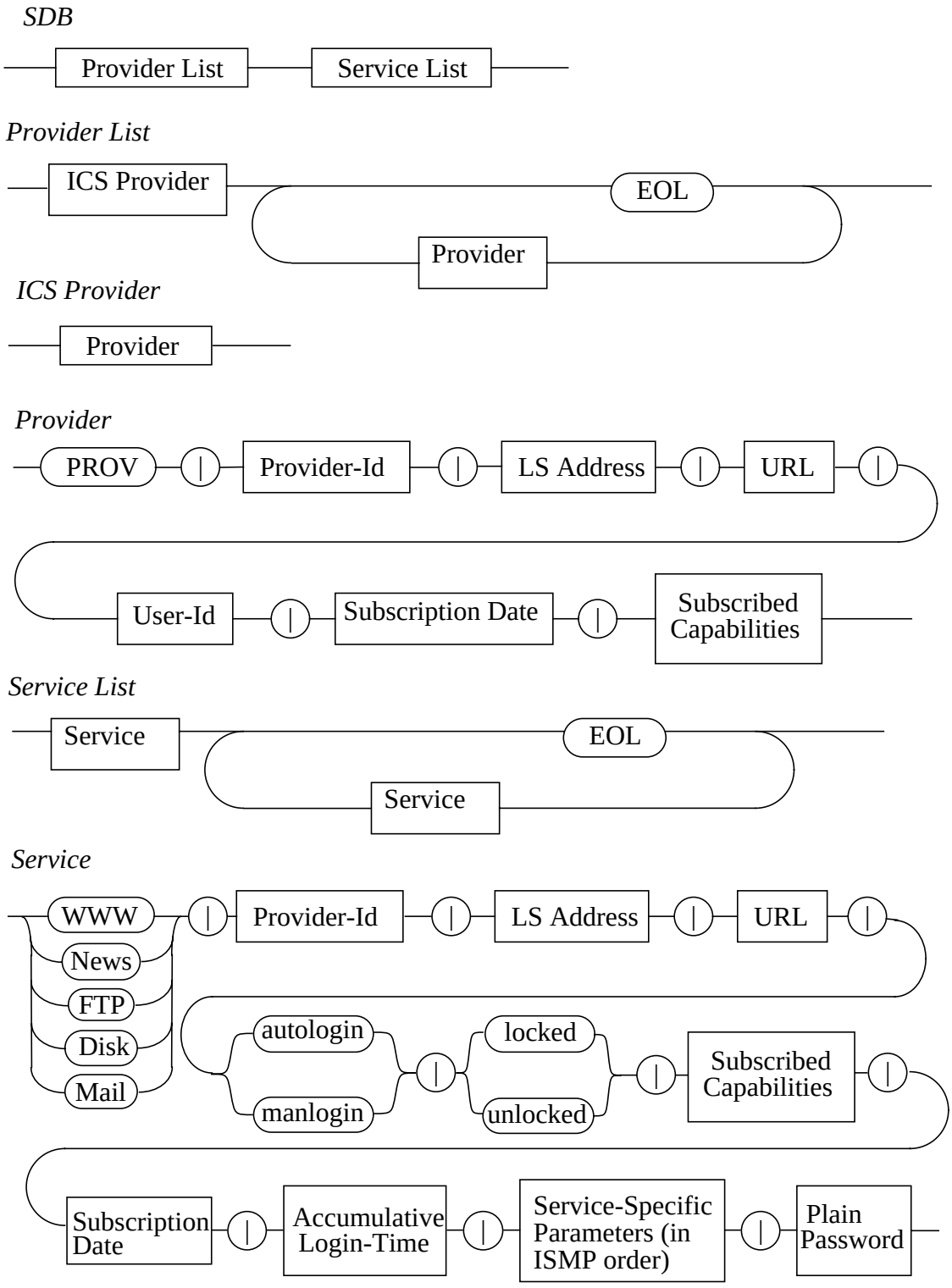

Figure 16: Syntax Diagram of SDB

3.1.5 NFS UID Mapping

When one subscribes at a provider, a unique user-name and user-id is assigned to the subscribing user. He or she can use them for accessing all other services within the provider. Furthermore, the user-id assigned by the ICS provider is used as a local-id at the network computer after a successful connection is established (uid “a” in Figure 17).

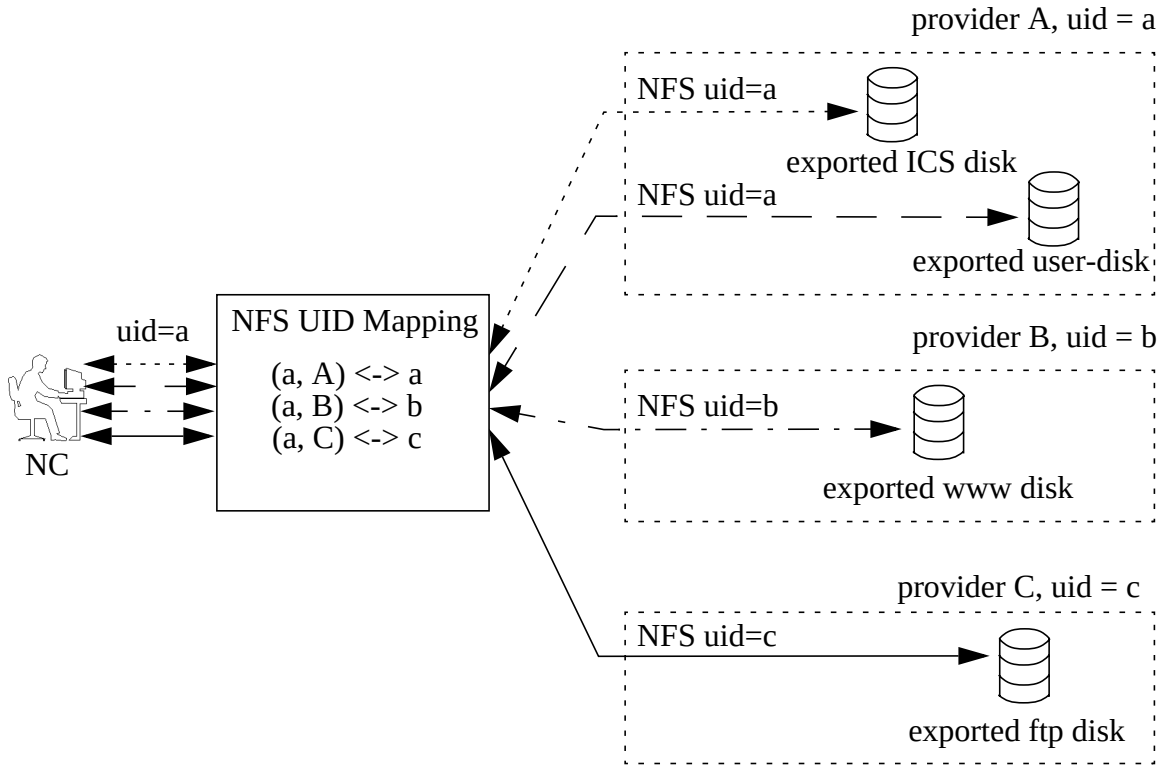

Figure 17: NFS UID Mapping

The concept described in this document allows the user to subscribe to services at various providers. This implies having as many user-ids for an NC user as the number of providers the user is subscribed to.

This is no, problem as long as the user does not subscribe to an NFS-disk-place at various providers. (Note, that disk place is exported over the NFS to users of “ICS”, “disk”, “www” and “ftp” services.) If he does so, his or her local user-id (assigned by the ICS server) will not match the user-ids at other providers, and it becomes impossible for him/her to access his/her NFS servers with the correct user-identity and user-rights.

Another problem is that the control software, the NC Session Manager (NCSM), runs under a super-user id which is mapped to the user-id “nobody”, which has minimal access rights at the NFS server. Therefore, the NCSM cannot access any user-specific service information stored in the *service database (SDB)* located at ICS server, among other things.

The solution is to map the local user and superuser ids to the user-ids at other providers and vice versa. The mapping should take place in the NFS client code (i.e. in OS-kernel[6]) and be set up with a shell utility.[7] The scheme is illustrated in Figure 17.

Note: Another place to map user-ids to is the NFS server; for example the Linux user-level NFS server is able to map the user-ids of all the NFS requests coming from an IP address to a predefined id. (The options 'anonymous' and 'squizeall' must be set.)

3.1.6 IP Masquerading

IP masquerading is a way to hide multiple network hosts behind a single IP address. This is done by many network providers to save their limited IP address space. It is based on mapping the pair {dummy IP address of a *masqueraded host*, port number} to the pair {IP address of a *masquerading host (MH)*, port number} and vice versa. The network sees the masqueraded host as the MH and its own address is unknown on the net!

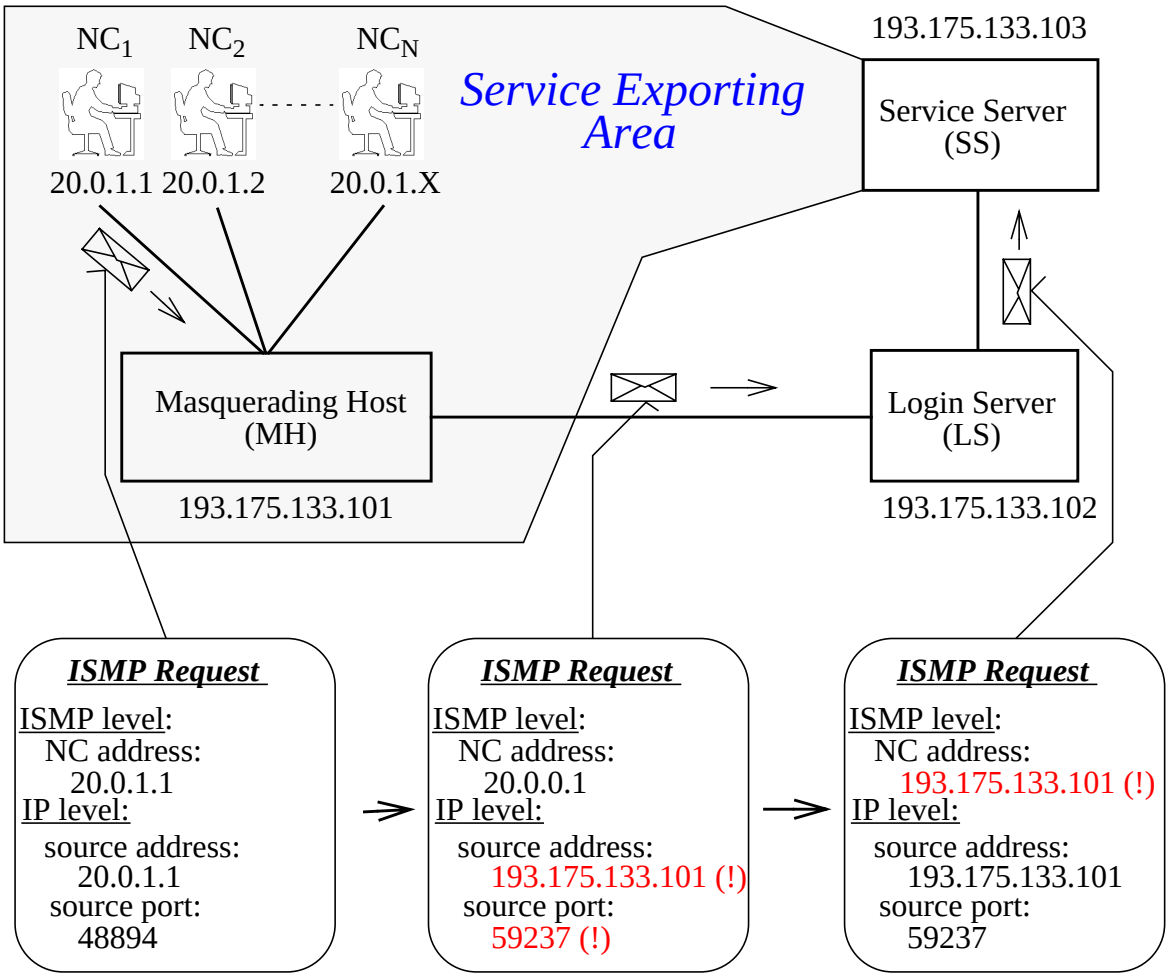

Figure 18: IP Masquerading

6. In our implementation, the kernel XDR level of NFS client has been patched.
7. The *mount* utility has been patched in our implementation.

Note: IP masquerading is also known as Network Address Translation (NAT), see [rfc1631] for more details.

The IP masquerading has several consequences for the network computer:

- The LS must replace the ISMP field “NC address” with the source IP address of the incoming ISMP connection before passing on the ISMP requests (see Figure 18).
- Then, servers will correctly export services to the MH specified in the ISMP “NC address” field. Unfortunately, this introduces lower security, because it makes services available to *all* of the masqueraded hosts behind the MH, as well as to the MH itself. (See the “Service Exporting Area” in Figure 18.)
- Accessing servers at an NC is not possible when masquerading — the well-known server ports of the NC *cannot* be reached on the network over the MH. In general, the NC works as a networking client, but there are few services which might be quite useful: ident, finger, etc. Also if the “application service” does not only use WWW and Java applets, but also uses “X Window System Protocol” [rfc1013], as X terminals do, then masquerading is not possible. (The X server would be unreachable over the network.)

The conclusion is that IP-masquerading has to be used carefully — lower security and the unreachability of the servers running on an NC has to be taken in account.

3.2 The NC Session Manager (NCSM)

The NCSM is a GUI-equipped application responsible for subscribing new services and managing subscribed ones. NC users manage all their services over with this central application. The NCSM’s main tasks are

- establishing and disconnecting PPP connections with network providers,
- subscribing new services with the ISMP,
- keeping track of subscribed services, unsubscribing, logging them in/out and changing their parameters with the ISMP,
- starting client applications for accessing subscribed services,
- displaying charging information,
- watching the ISDN connection, dropping it during idle periods, and
- creating a secure environment which cannot be damaged by hostile applications.

This chapter addresses portability of the NCSM, NCSM’s control flow and process relationships.

3.2.1 Portability of NCSM

One of the goals of the NC-project is to create an *easy-to-port* test platform. Today’s most sig-

nificant Internet-oriented, platform-independent programming language is Java. Program code written in the Java language can be run on any platform which has a Java interpreter. Java computing has been endorsed by all major hardware and software vendors in the world and the Java Virtual Machine (JVM) is available across all widely-spread operating systems and computer architectures. It has been proposed that the NC Session Manager (NCSM) be programmed in the Java language.

However, there are certain OS-related functions performed by the NCSM which are platform dependent (e.g. mounting filesystems, starting sub-processes, creating a chroot-ed filesystem environment, etc.) and therefore cannot be programmed directly in the architecturally neutral Java. There are two ways to unite the Java-code with these functions: the functions can either be written and compiled as *native methods* (e.g. in the C language) and linked to a Java class library, or they can be made available as stand-alone utilities which can be started as a sub-process by the JVM.

The former alternative is definitely the faster. The latter one is more portable and easier to administer: The utilities can be written in a Bourne-shell in a POSIX-compliant fashion and effortlessly ported to any UNIX-like platform[8]. The shell-scripts can be easily modified, even by administrators. And last but not least: A lot of programming and debugging work can be avoided when one uses ready-to-use shell utilities.

We prefer these features in spite of the loss of performance. Therefore the NCSM is designed as a Java application which calls external Bourne-shell scripts[9]. The loss of performance (~ tenths of seconds, according to measurements described in Section 3.2.1.1) can be neglected. However, future rapid commercial implementations can also call the native methods.

3.2.1.1 The Performance Evaluation of the Portability Model

This section examines the overhead introduced by the usage of Bourne-shell scripts. The overhead is caused especially by the following actions:

- the NCSM forking and executing the Bourne-shell
- interpreting shell-commands
- the shell forking and executing the shell-utilities

A pair of test suites (see Figures 19-22 for their source codes) has been defined to estimate the resulting overhead. Both suites differ only in their implementation (native method vs. Bourne-shell-calls), the steps in the subroutines are the following typical actions: (1) changing a directory, (2) writing to a file, (3) starting a subprocess and waiting for its termination, (4) mounting

8. Bourne Again Shell (BASH) is also available for MS-DOS.
9. All NCSM execution calls are separated in the SystemCommands class.

and unmounting a filesystem, (5) creating and removing a directory. The suites have been run 1,000 times to achieve satisfactory precision. The testing device was a PC with Pentium processor (79 MHz), 40 MB RAM and Linux 2.0.30. The test results are summarized in Table 11.

Table 11: Test Suite Results

	native method calls	Bourne-shell calls
total time [ms]	130,060	233,319
time per suite [s]	0.130	0.233
relative time	100%	179%

```
#!/bin/sh -e

# step one: change directory
cd /tmp

# step two: write to a file
echo “Fantomas was here” > testFile

# step three: start a subprocess and wait for it
/bin/date

# step four: mount and unmount a directory
mount -t vfat /dev/hda1 /mnt
umount /mnt

# step five: create and remove directory
mkdir /tmp/tstdir
rmdir /tmp/tstdir
```

Figure 19: The Bourne-shell Part of the Code for the “Bourne-Shell” Test-sequence

```
import java.util.Date;

class TestShell {

  static Runtime rt = Runtime.getRuntime();

  static public void main( String[] args )
  {
    System.out.println("Shell-tester started");
    Date d1 = new Date();
    test();
    Date d2 = new Date();
    System.out.println("Shell-tester finished");
    long tt = d2.getTime() - d1.getTime();
    System.out.println("Total time [ms]:"+tt);
  }

  static void test() {
    Process p;
    for (int i=0; i<1000; i++) {
       try {
            p = rt.exec( "/net/u/nthp/jku/test/test.sh" );
            p.waitFor();
       } catch( Exception e ) {
            System.out.println("Execution failed");
       }
    }
  }

}
```

Figure 20: The Java Part of the Code for the "Bourne-shell" Test-sequence

```
import java.util.Date;

class TestNative {

  static { System.loadLibrary("test"); }

  void run() {
    for (int i=0; i<1000; i++) oneTest();
  }

  native void oneTest();
}
```

Figure 21: The "Java" Part of the Code for the "native" Test-sequence

```
#include <jni.h>
#include “TestNative.h”

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mount.h>
#include <linux/fs.h>

#define TSTDIR “/tmp/tstdir”
#define MOUNTDEV “/dev/hda1”

extern int errno;

JNIEXPORT void JNICALL Java_TestNative_oneTest
  (JNIEnv *env, jobject job )
{

  int fh, rc, status ;

  /* step one: change directory */
  rc = chdir( “/tmp” );

  /* step two: write to a file */
  rc = fh = open( “testFile”, O_WRONLY | O_CREAT | O_TRUNC );
  rc = write( fh, “Fantomas was here\n”, 17 );
  rc = fchmod( fh, S_IWUSR | S_IRUSR );
  rc = close( fh );

  /* step three: start a subprocess and wait for it */
  rc = fork();
  if ( rc == 0 ) /* child */ execl( “/bin/date”, “/bin/date”, NULL );
    else /* parent */ wait4(  rc, NULL, 0, NULL );

  /* step four: mount and unmount a directory */
  rc = mount(MOUNTDEV, “/mnt”, “vfat”, MS_MGC_VAL, NULL );
  rc = umount(MOUNTDEV);

  /* step five: create and remove directory */
  rc = mkdir( TSTDIR, 0 );
  rc = rmdir( TSTDIR );
}
```

Figure 22: The “native-C” Part of the Code for the “native” Test-sequence

3.2.2 Definition of NCSM's Control Flow

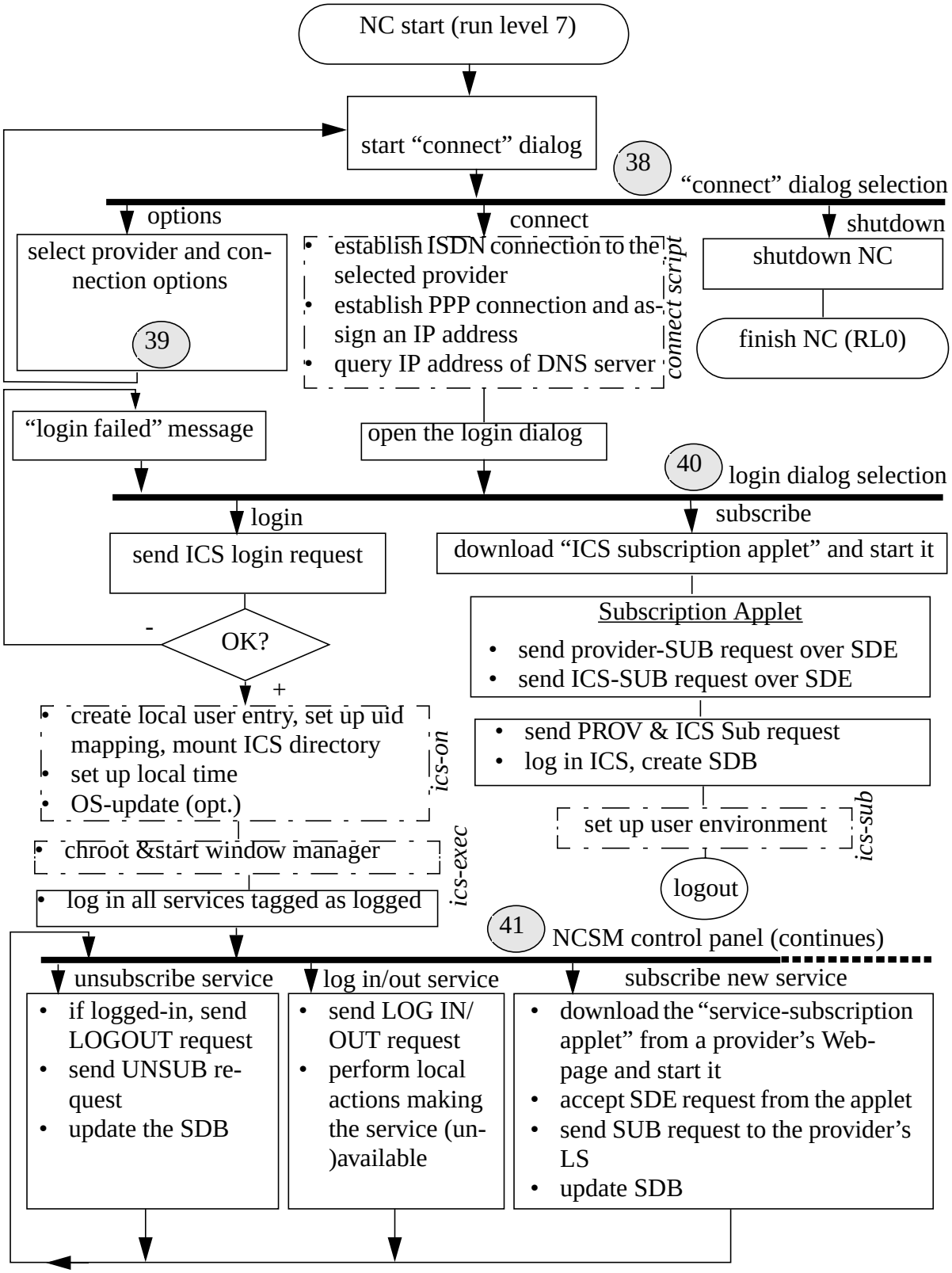

Figure 23: NCSM Control Flow Diagram - Beginning

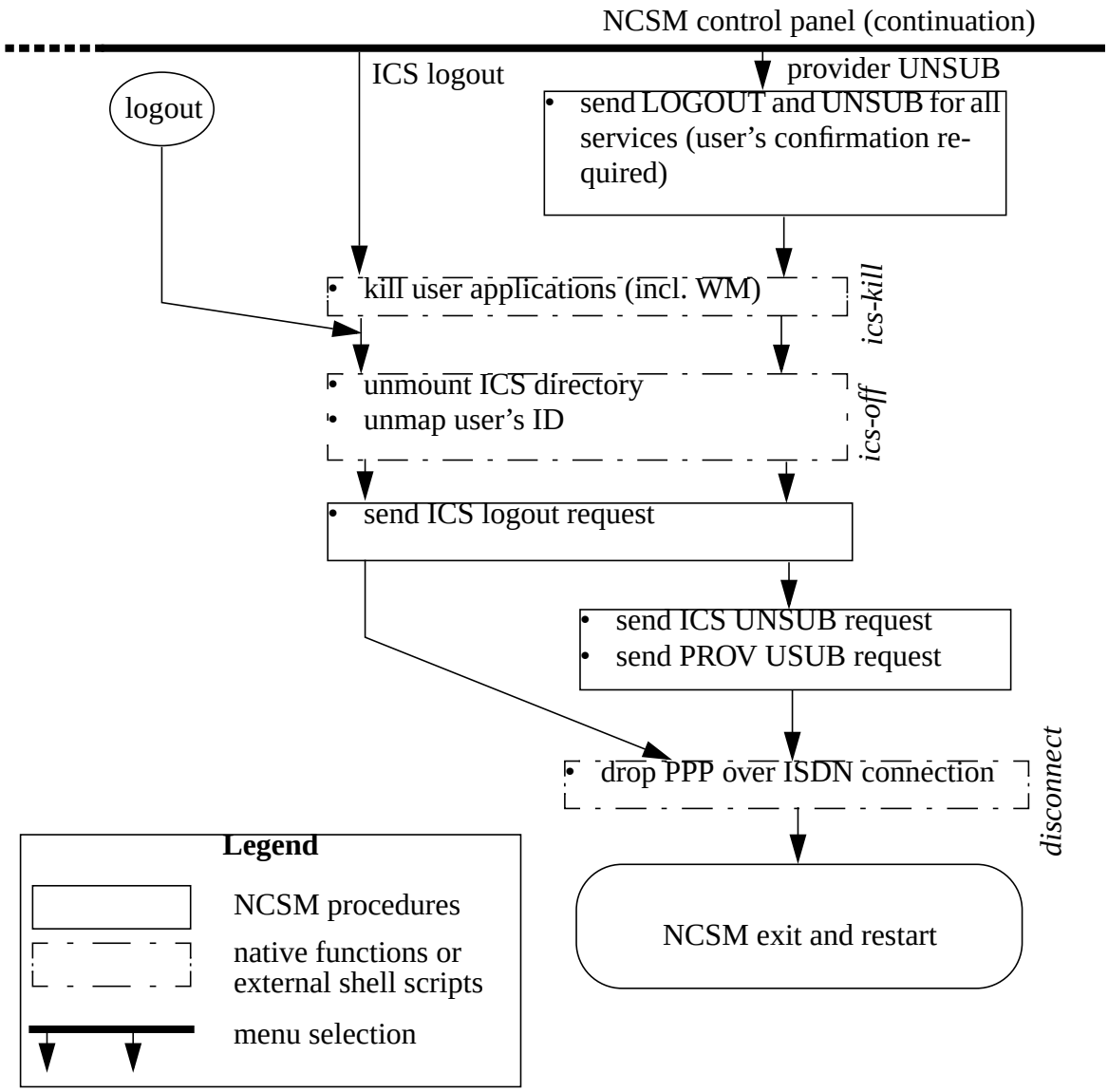

Figure 24: NCSM Control Flow Diagram - Continuation

NCSM's control flow is defined in Figures 23 and 24. The numbers corresponding to figures with examples of implementation of user-interface (Appendix A) are referred to in small shaded circles.

3.2.3 NCSM Process Relationships

In a hierarchical process management system, NCSM is a subprocess of its parent-process and may itself start its own subprocesses (also called child-processes).

How to start the NCSM is more or less a matter of implementation. Unix-like implementations may use our scheme in which the NCSM is started by X Window *xinit* which is in turn started

by the UNIX process number 1, *init* (Figure 25, see Section 4.3.2.1 on page 80 for more details).

NCSM's subprocesses are more interesting. There are two types: *NCSM control subprocesses* and *user applications*. The former ones are those discussed in the Chapter "Portability of NCSM" on page 47. They perform OS-specific functions and are important to session control — they establish ISDN connections, mount remote disks, set up the system's real-time clock, and so on. They always run under a superuser id, so that they can modify system resources, and the NCSM always watches carefully for their terminations. Their names, starting time and functionality are defined in Figure 23.

The *user applications* are processes launched by the user. These are mainly applications for accessing network services: file manager for accessing a disk-service, news reader for accessing a news service, etc. They can be started either directly by the NCSM or by any of the running *user applications.* (The window manager is also a user application which is started by the NCSM![10]) In order to ensure a secure environment, all of them are run exclusively under the user's local id in a *separate subfilesystem*[11]. They are not allowed to access any directories outside of this subfilesystem. Moreover, this subfilesystem is dynamically created at the beginning of each session[12], and even if it is damaged either because the user made an error or because of an intruder's attack, it appears again in the next session undamaged.

The user applications which access a service have to be killed before the service is logged out. The reason for this is that it may be impossible to stop providing a service which is still occupied by an application. A typical example is the disk service: a remote directory can not be unmounted until an application accesses it. Therefore, the NCSM starts user applications in special process groups[13], keeps track of them and kills them on service log-out. The user applications are free to create their own subprocesses, but it is assumed that they will not create new process groups thus giving the NCSM a chance to follow them[14].

The process structures for a common NC session and for an ICS-subscription session are shown in Figures 25 and 26. The numbers of the figures which show examples of the implementation of a user-interface (Appendix A) are referred to in small shaded circles.

10. WM is started from script "ics-exec"
11. The path is currently defined as /usr/nc in *init* script *rc.nc*; utilities chroot and su are used, see Section 4.3.2.2.
12. The "chroot" utility is used for separation of a part of a filesystem. Read-only subdirectories are imported in the subfilesystem with the "/opt/FOKUSnc/bin/mountlocal" script, other subdirectories are restored in the init-script "/opt/FOKUSnc/etc/rc.nc.d" from a package "/opt/FOKUSnc/etc/usrnc.tar".
13. The utility "/opt/FOKUSnc/bin/spgrp" has been written for this purpose. spgrp creates a new process group, records its id in a file and executes an application.
14. This is the reason why Open Look window manager (olwm) can not be used — applications launched by olwm are started in a new own process group.

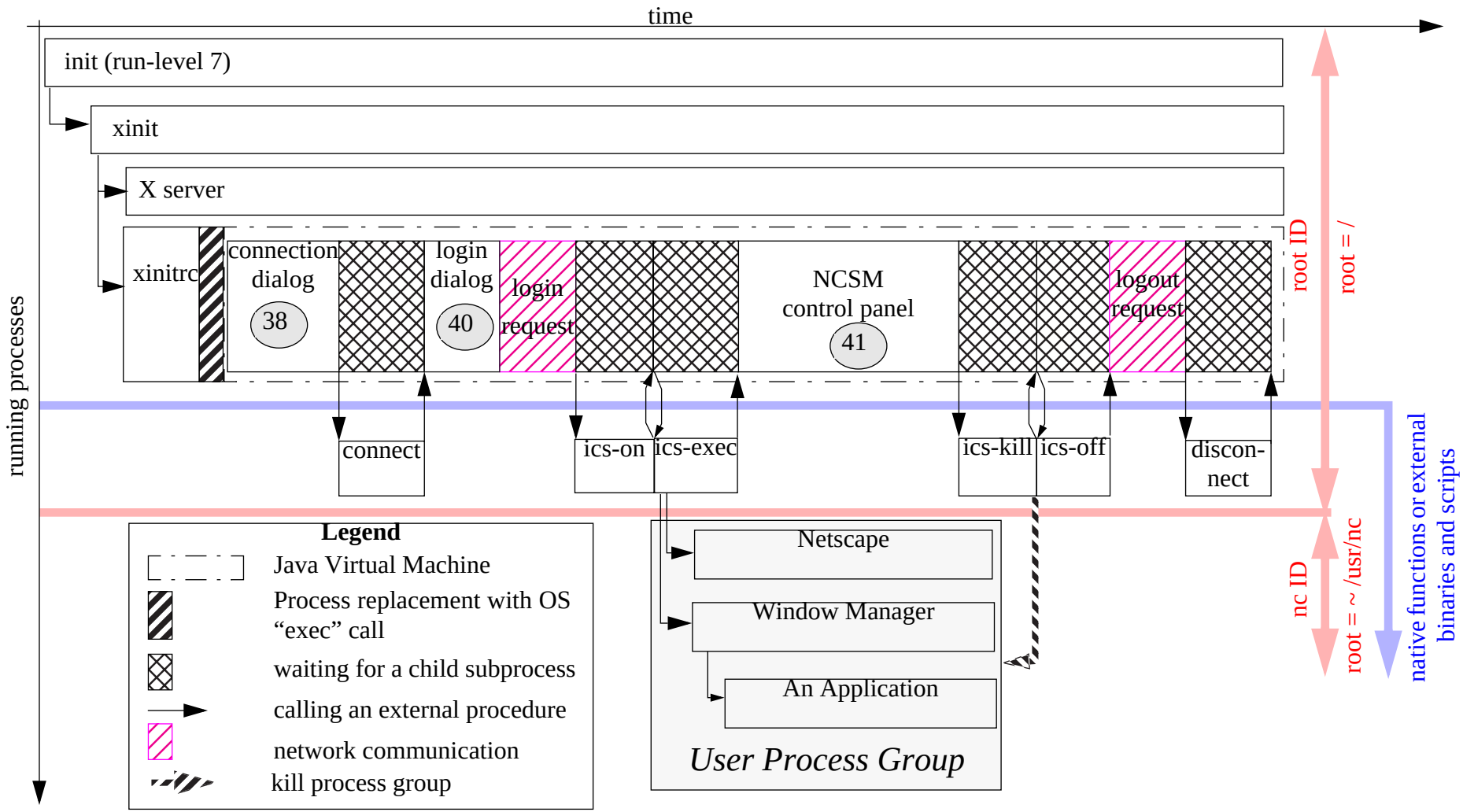

Figure 25: Process Relationships of an NC Session

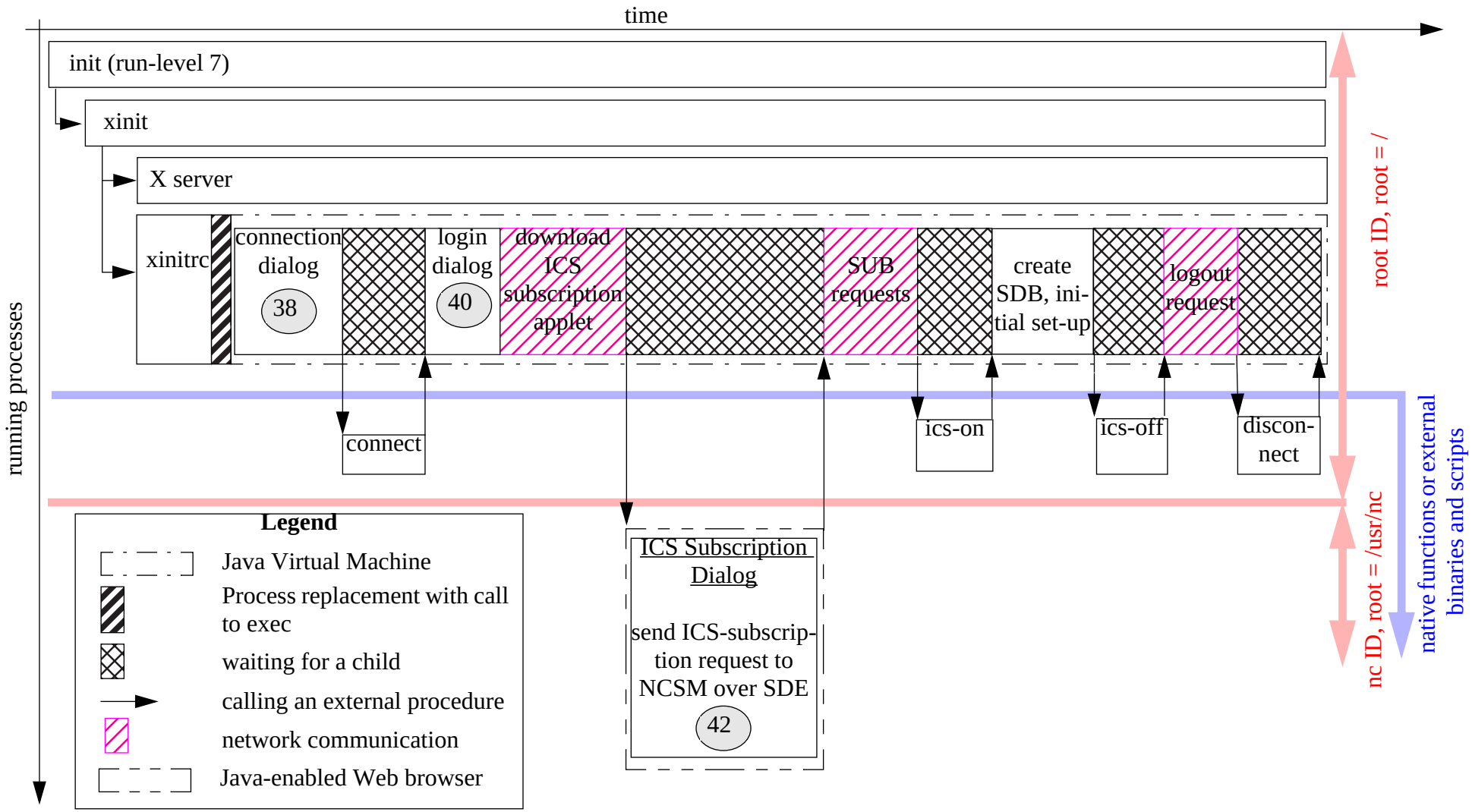

Figure 26: Process Relationships while Subscribing to an ICS

3.3 The Service Subscription

Not only in the real world, but also in our NC architecture do users have to subscribe to services before they are allowed to access them. The objective is to perform all of these steps purely electronically, without using any of the bureaucratic methods that have been used in the past. Users are to just buy an NC and plug it into power supply and an ISDN socket. Our ISMP-based method exploits the WWW and signed applets. There are three ISMP subscriptions a user must perform in order to be able to use a new service:

- He or she must be known to service-provider; the process of identification of a user at a provider is called *provider subscription.*
- He or she must subscribe to *Internet Connectivity Service (ICS)*; otherwise they do not have access to any services provided in Internet.
- Identified users with subscribed Internet Connectivity Service can subscribe to any additional services with the required parameters, such as a 100 MB disk service.

According to ISMP definition, the NC client, the NCSM, is supposed to send an ISMP subscription request to the LS and receive an ISMP response. So it is the NCSM's task to ask users for all necessary subscription data before mapping them into an ISMP request. So far so good. This solution should work perfectly. But what about a provider who wishes to learn additional information about the subscriber, let's say his date of birth? The place for additional parameters is predefined in the ISMP (The subscription request can take any number of provider-specific parameters at the end of the PDU, see the Table 2 on page 18.), but the provider's wishes are unknown to the NCSM — the hard-wired NCSM user interface does not allow the poor provider to customize the subscription form.

The idea is to subscribe over WWW-pages which are administered and freely-customized by the provider and to equip the NCSM with a back-door for communication with a "subscription applet (SA)" run within a Web-browser. (We call this "back-door" *SDE - Subscription Data Exchange*). The WWW pages may comprise any provider's most up-to-date commercial information and may ask the subscriber for any additional information specifically required by the provider.

The question is how to design the SDE. In general, any type of one-way interprocess communication may be used: FIFOs, shared memory, TCP, UDP, shared files, Microsoft DDE, etc. The UDP has been selected for the SDE[15] — it is architecturally neutral and has a Java API. (The SDE takes place on the local host. It is therefore not necessary to use reliability any of the features of the TCP.) The entire subscription communication flow is depicted in Figure 27.

15. Implemented in class SubscriptionThread. The subscription daemon listens on port 9010 (Subscription.SDE_PORT).

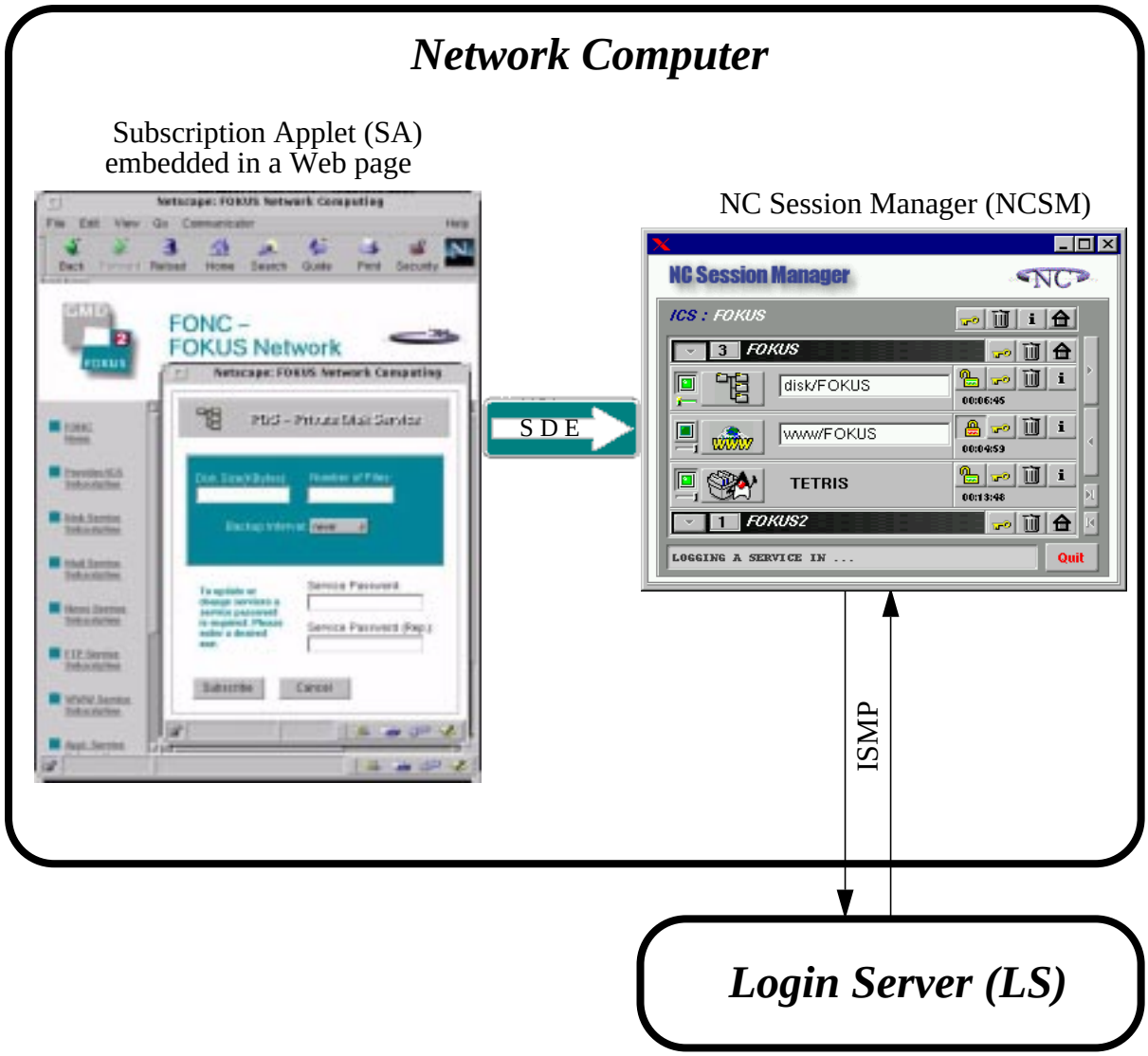

Figure 27: Subscription Communication Flow

3.3.1 SDE Security

The subscription concept is open in order to enable the providers to individually customize their subscription interfaces. The SDE, the “back door” to the NCSM, can unfortunately become a target of security attacks. Security policy is defined as follows in order to minimize all risks:

- An incoming SDE request is accepted only if its source is the local host. Otherwise it is immediately rejected and a warning is displayed[16].
- The WWW browser (Netscape), which has support for the signed applets and control of the local resources accessed by an applet, is used. Only the applets which are trusted by the NC-

16. Implemented in SubscriptionThread.readNewRequest().

user are allowed to send SDE requests. See [osr] for more details.

- SDE allows *one-way* communication only. This prevents hostile applets from receiving any information from the NCSM.
- The NCSM checks[17] the service-passwords proposed in an SDE request. If they are easy to guess, then no subscription is initiated and a warning is displayed. The username proposed in the provider-subscription request is also checked[18] — some characters in a username, "+" for example, have a special meaning in an NIS database and may not be used.

3.3.2 Subscription Steps

The subscription architecture and security policy require the following steps for a successful service subscription:

- The user downloads the provider's WWW subscription page which has an embedded signed *subscription applet (SA).*
- The user specifies his desired service parameters and initiates the subscription; he or she confirms that the SA may use the SDE.
- The SA sends the subscription request to the NCSM over the SDE.[19]
- The NCSM checks the request. If the request is correct (i.e. if it comes from the local host, its syntax is correct and its proposed password is not easy to guess) it is forwarded to the LS over the ISMP.
- The LS performs the service subscription and sends a reply to the NCSM.
- The NCSM adds the subscribed service to the Service Database (SDB)[20]; no feedback is sent to the SA.

3.3.3 Specifics of ICS Subscription

Because the SDB is tightly coupled with the ICS (see Section 3.1.4 on page 43), a new user cannot subscribe to a provider or store any subscription information if he does not have ICS-access to the SDB. He also cannot subscribe to the ICS until he is subscribed at a provider. Therefore, an ICS subscription request is integrated with a provider subscription request whose parameters are passed along at the SDE level. The NCSM performs the following steps during an ICS subscription:

- The integrated PROV-ICS SDE request is sent from a subscription applet to the NCSM.

17. Implemented in SubscriptionThread.checkPassword().
18. Implemented in SubscriptionThread.checkUsername.
19. A subscription applets need not implement SDE. SDE interface (methods subscribeProvider, subscribe-ICS and subscribeService) is available in class Subscription.
20. Implemented in NCSession.subscribe.

- The ISMP provider-subscription request is sent to the login server and subscription information is temporarily cached.
- The ICS is subscribed.
- The SDB is initialized with the following cached provider-subscription information:
  - The ICS is automatically logged in[21].
  - The SDB is created, provider's record is put in.
  - The user's initial environment is set up[22].
  - The ICS is finally logged out.

Another point is that the ICS subscription applet (SA), unlike applets for subscription to other services, absolutely must be located in the Freely Accessible Area (see Figure 14 on page 38). The reason is obvious: An ICS cannot be subscribed over any Web-pages located outside of the FAA until Internet connectivity is enabled.

3.3.4 SDE Format

The format of SDE is intentionally made similar to the format of ISMP and SDB. This makes mapping an SDE request to an ISMP request easy. The SDE syntax diagram is shown in Figure 28.

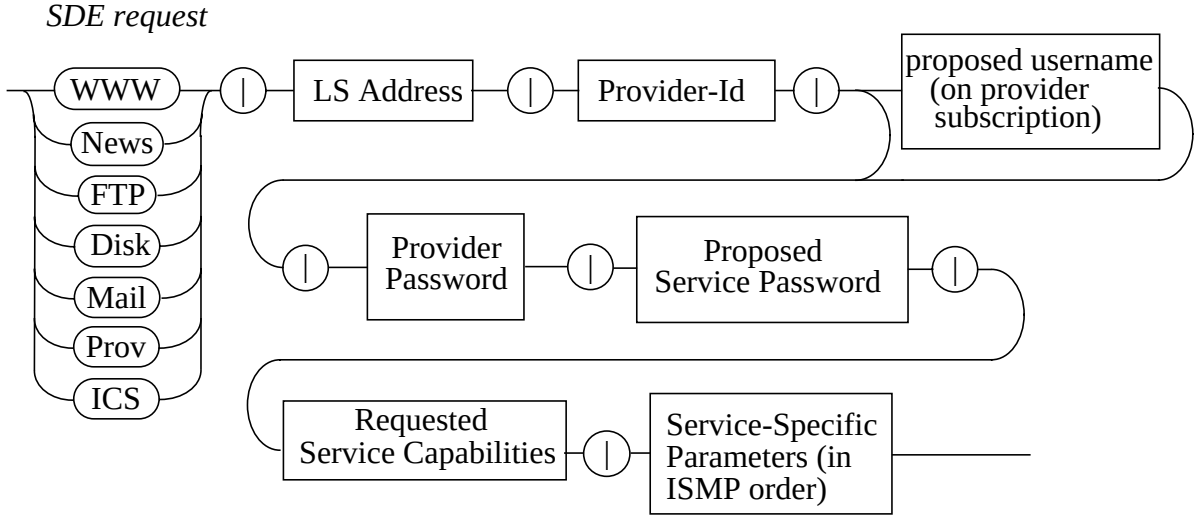

Figure 28: Format of an SDE Request

Note that the last SDE field is comprised by service-specific subscription parameters according to Table 2 on page 18. In particular, any number of additional provider-specific parameters can be requested at provider subscription.

21. This and the following steps take place in ICSService.afterSubscription().
22. Setting-up the initial environment is implemented in the script "ics-sub"; currently, Netscape preferences are set-up in this script.

3.4 The Update of Operating System

Today's NCs are defined as diskless devices. All software, including operating systems, is downloaded over network. This might be reasonable in LANs, but too long of transmission times and too high of connection expenses make this approach impossible in the WAN area.

This is the reason why we propose to equip the NC with a *low-capacity storage* for *caching network data*, especially operating systems (OS). This method spares network bandwidth but requires additional features: version control and a reliable means for updating the OS.

The version control is built-in in ISMP and NCSM. The NCSM receives the number corresponding to the most recent version of the OS available[23] over ISMP at every ics-login. Afterwards, this version is compared[24] with the version of the locally cached OS[25]. If the local version is out-of-date, the OS is updated automatically. An OS update consists of the following steps[26]:

- mounting an NFS filesystem with the up-to-date OS version,
- downloading and uncompressing the OS in a reserved partition,
- unmounting the OS filesystem and logging out the ICS,
- disconnecting ISDN, and
- rebooting the NC from the partition with the new OS.

Note that this procedure requires two cache partitions: one for the current OS version and another one for the downloaded version. This guarantees that the NC can be rebooted from the old version in case of an emergency. If the OS update finishes successfully, the roles of both partitions will be swapped.

3.5 Open Issues

There are some important issues which have not been discussed yet. These are namely "*travelling data*", "*superservice*", *accounting* and *security*.

3.5.1 "Travelling Services"

The FOKUS concept supports enhanced mobility. NC users can access their environments from anywhere in the world. However, accessing German services from Australia over the Internet may take too long and accessing them directly over an ISDN channel would be extremely ex-

23. The OS version field is parsed in ICSService.parseLoginSpecificInfo().
24. The comparison takes place in the method ICSService.osUpdate().
25. The version of locally cached OS is saved in the file "/.NCOS.version".
26. Implemented in ICSService.osUpdate(); also the external script "ncosupd" is called by this method.

pensive.

The solution to this problem is to allow the subscribed services to “travel” with their users. NFS data, the mailing spool, applications and other resources would be moved towards the user as far as possible upon his request.

This feature, together with other mobility aspects, will be addressed in the next project stages.

3.5.2 “Superservice”

The goal of the FOKUS concept was to design a management protocol supporting all possible services. Today’s most used Internet services (Application, E-mail, News, NFS, WWW, FTP) are defined in ISMP.

However, there might appear new services in the future, such as Internet-telephony or video-on-demand. A static possibility is to extend ISMP to support the new services. More advanced techniques are proposed in this subchapter to support these without any need to extend existing protocol implementations.

One way to do this is to encapsulate the new service in the “application service”. For the sake of better comprehension, the usage of an encapsulated service is explained in the following example:

> *A provider decides to provide video on demand over the Internet. He designs a Java control application for accessing the service and offers it to his customers. The application may allow its users to specify session-specific parameters, for example the title of a desired movie, the frame rate, the resolution and other “Quality of Service” (QoS) parameters.*
>
> *An end-user who wants to watch a movie has to subscribe to the control application. After having logged in to the subscribed application, he or she is permitted to start it. The application asks him for the name of the movie and the required resolution. The playing of the video stream is started and continued until the user logs out from the service.*

Another possibility is to define an adaptive protocol mechanism for learning new services. (This is what I call *superservice.*) The NC would have to query an LS about all of the properties and actions related to the new service before using it. This way is very flexible but requires additional security enhancements — the downloaded unverified “service-actions” might become a target of an intruder’s attack on the NC’s security. This would be especially dangerous for service actions requiring access to the NC’s local resources. For instance, the *login action* of a *remote printer service* may want to set up the remote device in the NC’s printer database, which requires root privileges.

3.5.3 Multimedia Applications

The ISMP concept manages subscription and access control of today's Internet services. These data-oriented services rely on the Internet's "best effort" quality of data transmission.

Today, multimedia services are gaining popularity. The problem is that these services require higher quality of data delivery than the data-oriented ones. Parameters such as minimum bandwidth, maximum packet delay and peak rate must be specified to deliver such services.

The ISMP is not concerned with these issues. The multimedia applications are the same as any other from the ISMP's point of view and it is up to the applications, which provide an NC-user with a multimedia service, to ensure satisfactory delivery of the service (see the example in the previous chapter). To name at least one Internet technology the multimedia applications may use: The Real-time Transport Protocol (RTP) [rtp, rfc1889] as a transport protocol and Reservation Protocol (RSVP) [rsvp, rfc1889] to ensure the required bandwidth. Another system for negotiation, monitoring and the adaptation of QoS has been developed at the University of Montreal [qosna]. This technology allows multimedia applications to negotiate QoS and dynamically adapt to QoS variations in the system components and to changing user requirements.

3.5.4 Application Service

Not all of the details of application service have been discussed yet.

One objection to application service is that the rented Java application may remain in the NC's cache. This would make usage and charging control impossible for provider. A satisfactory solution would be to involve a dynamically-controlled license manager.

Another objection is that downloading a complex application may be very bandwidth-consuming. Downloading needed parts of code on demand and caching them might be utilized to minimize all unnecessary transfers and spare the bandwidth. Part of the application processing can take place at the server side to minimize the data traffic.

3.5.5 Accounting

Accounting is important for commercial deployment. At this project stage all accounting-relevant information, such as login-times, subscription-times, usage-extent and so on, are saved in a server's database. A database API for accessing these pieces of information is defined. An accounting module implemented by the provider, which contains the model he uses for charging customers, is supposed to use this API when calculating usage-charges.

In order to enable an end-user to control his costs, some of the accounting data (current login time, accumulative login time, subscription date) are regularly transmitted to him over ISMP.

Additional provider-specific information (e.g. usage-price) can be sent in the “capabilities” protocol field. (See protocol definition in the Section 2.4 on page 30.)

3.5.6 Security

The NC open model provides its users with increased flexibility and mobility. These features are inherently a source of possible security leaks, which can be used by malicious users who misuse someone’s identity in order to access private data or other resources stored on the network.

The most vulnerable points are the ISMP protocol and the NFS protocol.

In the near future the security of the transaction-oriented ISMP can be improved with the same means that are widely used in the banking area. Secure Socket Layer [ssl] would be reasonable. Java implementation for this is available [ssljt].

The security of the NFS can be insured by using well-known security enhancements such as *Kerberos* [pus] or *Secure RPC* [pus].

4 Implementation Issues of NCSM

NCSM is Java-written, multithreaded, GUI-equipped, object-oriented, network software responsible for managing services leased over the Internet. This chapter describes the details of its implementation. Its object oriented design is introduced, and miscellaneous incompatibilities, bugs and implementation tricks are pointed out. No topics related to the general concept of internetwide network computing are discussed here.

4.1 Object-Oriented Design (OOD)

This chapter introduces the design patterns used in object-oriented implementation of NCSM. Mostly, they come from the famous book “Design Patterns” [dp]. The notation for diagrams in this section (Figure 29) is the same as that used in [dp], which is in turn based on OMT (Object Modeling Technique) [RBP+91, Rum94].

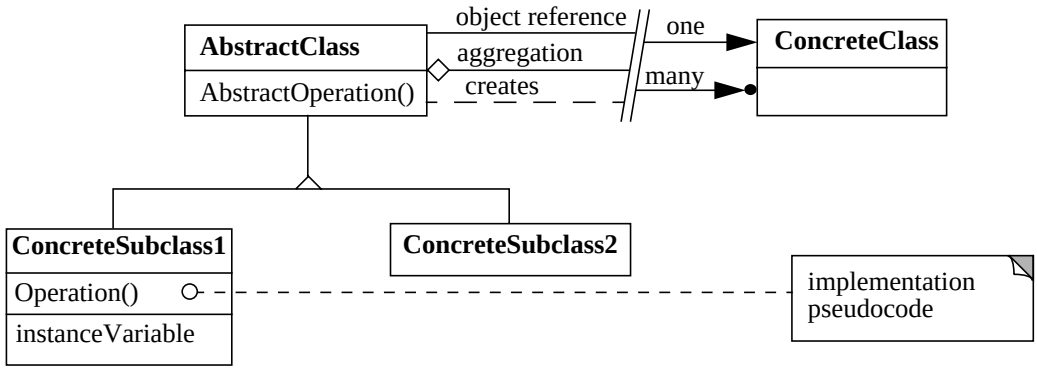

Figure 29: Class Diagram Notation

Note: object aggregation is actually not used because Java supports object referencing only.

4.1.1 NCSM State Machine

The NCSM is defined as a state machine according to Figure 30; transition details are described in Table 12. The state transitions are initiated at the user’s request via GUI. Allowed subscription possibilities with respect to the machine’s state are also tagged in the diagram. “The design idea is to introduce an abstract class *NCState* to represent NC’s state. The *NCState* class declares an interface common to all classes that represent different operational states. Subclasses of NCState implement state-specific behavior. The class NCMachine maintains a state object that represents the current state. The class NCMachine delegates all state-specific requests to this object. NCMachine uses its NCState subclass instance to perform operations particular to the state of the machine.” (quoted from [dp] with modified class names)

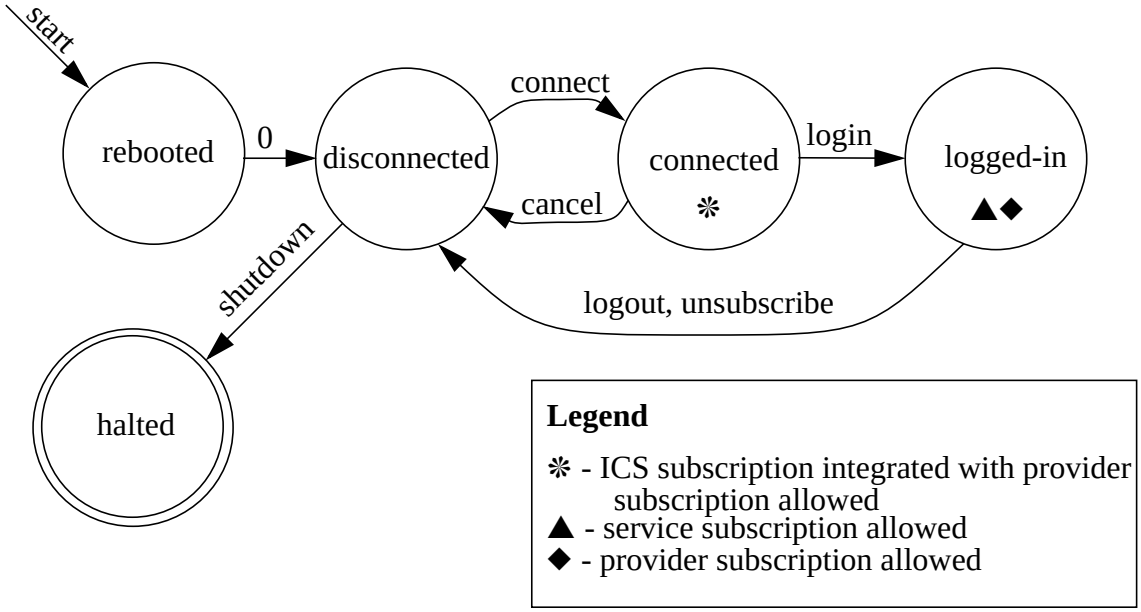

Figure 30: NCSM State Machine

.

Table 12: Transition Details

Originating State	Message	Transition Action	Destination State and its Initialization
connected	login	logging-in ICS (calling the 'ics-on' script)	**logged-in** NCSession.defaultAction (reading SDB, autoLogin, calling the 'ics-exec' script which starts WM and Net-scape)
	cancel	releasing the ISDN connec-tion (calling the 'disconnect' script)	**disconnected** no initialization
logged-in	logout	logging-out ICS (calling the 'ics-kill' and 'ics-off' scripts), releasing the ISDN connec-tion (calling the 'disconnect' script)	
	unsubscribe	logging out ICS (calling the 'ics-kill' and 'ics-off' scripts), unsubscription of ICS and provider, releasing the ISDN connection (calling the 'dis-connect' script)	

Table 12: Transition Details

Originating State	Message	Transition Action	Destination State and its Initialization
disconnected	connect	establishing ISDN connection (calling the 'connect' script)	**connected** no initialization
	shutdown	shutting the NC down	**halted** no initialization

Whenever a user initiates a change of the state machine, the NCMachine object changes the state object it uses. After the transition method[1] of the current state instance finishes processing, it creates and returns an instance of the next state (see the implementation pseudocode in Figure 31). Additionally, an object of *Base* subclass which is associated with the new state, is instantiated, and lastly its initialization method[2] is called. (If the user tries to initiate a transition which is undefined for the current state or an error condition occurs during the transition an exception is thrown and the machine's state remains unchanged.[3]) The *Base* subclasses represent detailed states' behavior, define all operations performable in the state and connect NCSM's code with graphic user interface (GUI). A GUI object is instantiated created by the method *startUI*[4]. It communicates with its creator, which is an instance of *Base* subclass, over a messaging mechanism (see Section 4.2 for more details). The complete class relationships are shown in Figure 31.

4.1.2 Services

The concept introduced in this document deals with *service management*. Therefore, the structures representing Internet services are of special importance. In our design, the abstract superclass *Service* defines features common to all services:

- ISMP operations (subscribe, unsubscribe, change, login, logout)[5],
- service state (see state diagram in Figure 2 on page 16)[6],
- accounting information[7],

1. Methods overriding NCState.{connect | login | shutdown | logout | cancel | unsubscribe}.
2. Base.init
3. Implemented in NCState.undefined, NCMachine.sendMessage.
4. 'connect dialog' (Figure 38) UI is instantiated in FNCConnect.startUI, 'login dialog' (Figure 40) is instantiated in FNCLogin.startUI and 'NCSM Control Panel' (Figure 41) is instantiated in FNCSession.startUI.
5. Service.{login | change | subscribe | unsubscribe | logout}
6. Service.isLoggedIn
7. Service.{set | get}{Current | Accumulative}LoginTime, Service.{get | set}SubscriptionDate, Service.getAccountingInformation

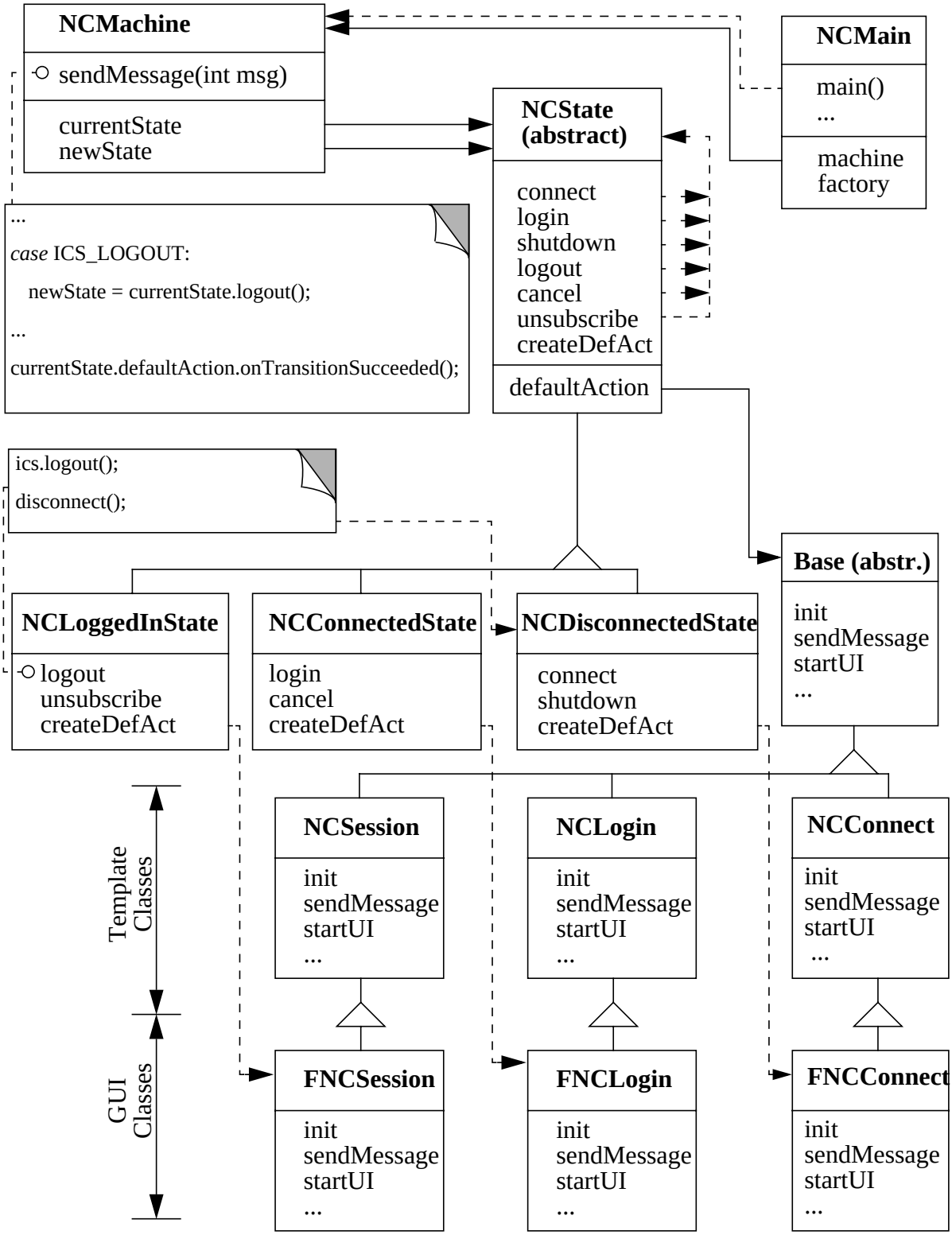

Figure 31: Class Diagram of NCSM's state machine

- SDB format[8],
- etc.

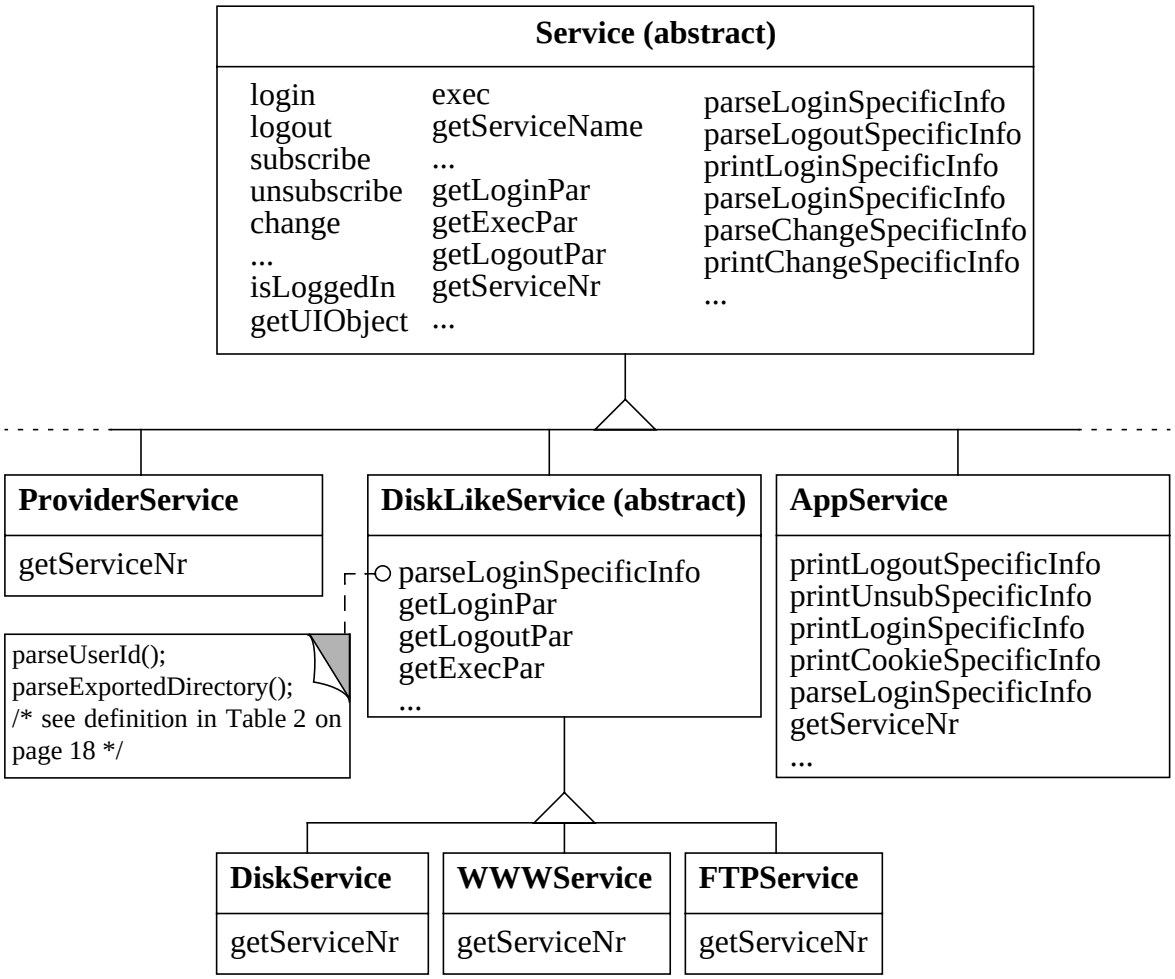

Figure 32: Service Class Hierarchy

Concrete subclasses (e.g. *AppService*) define behavior specific to concrete services. In particular, they define:

- the number and format of service-specific parameters for ISMP, SDB and SDE[9]
- service-specific actions, which shall take place after (un-) successful ISMP operations, and their parameters[10]
- the application associated with the service and the parameters passed to it,[11]

8. Service.{read | write}Config
9. Methods overriding Service.{print | parse}{Unsub | Sub | Login | Logout | Change}SpecificInfo, subclasses of inner class Service.ServiceParameters instantiated in overriden Service.getNewServiceParameters.
10. Methods overriding Service.after{Login | Change Subscribe}, Service.before{Logout | Unsubscribe}, Service.get{Login | Logout}Par; shell-scripts <service-id>-{on | off}.
11. Methods overriding Service.getExecPar, Service.exec; shell-script <service-id>-exec.

- ISMP service-id[12],
- etc.

The class hierarchy is shown in Figure 32.

4.1.3 ISMP Classes

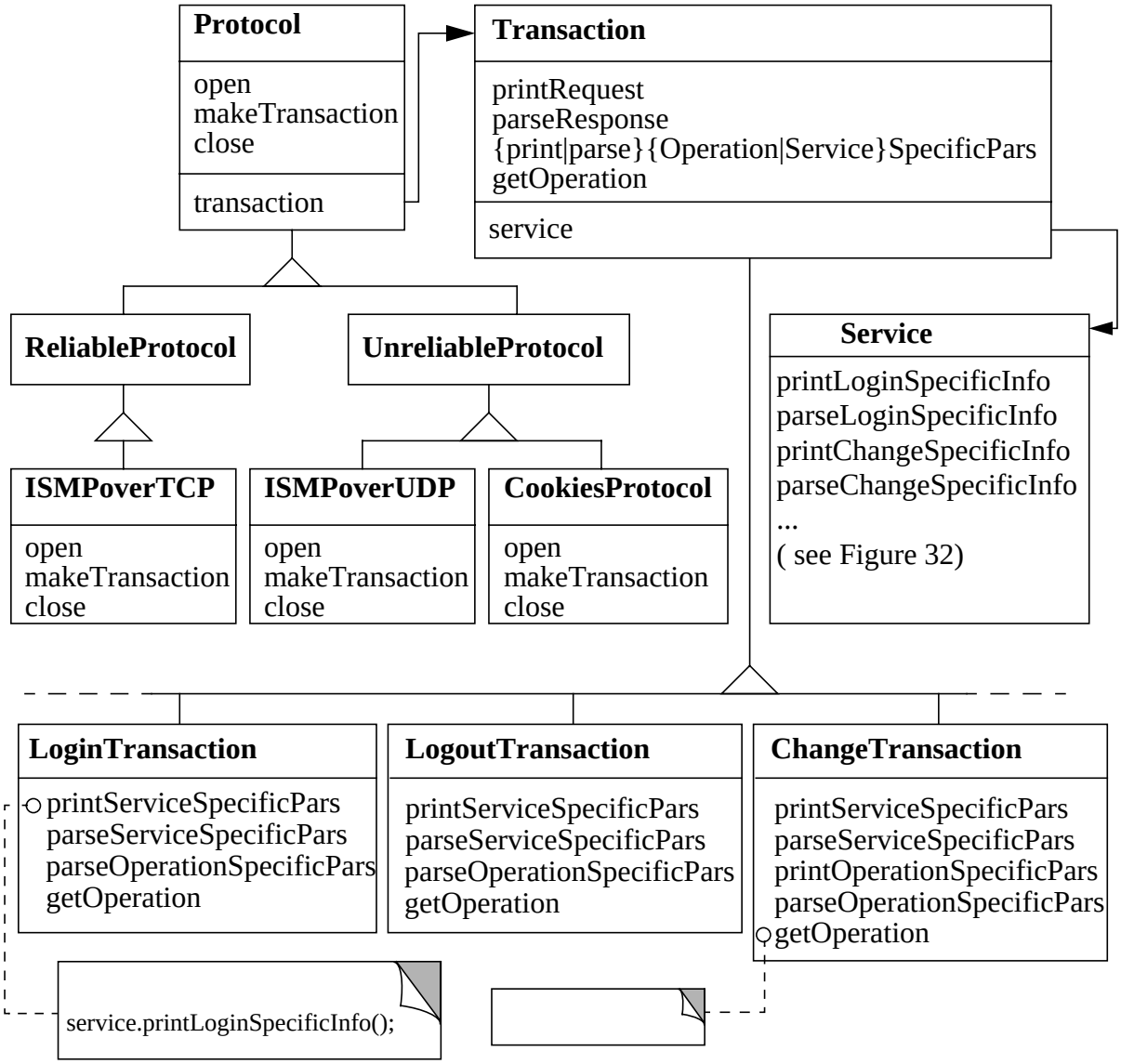

Figure 33: ISMP Class Hierarchy

The structure of ISMP classes strictly distinguishes between data and algorithms. Protocol algorithms (3-way handshake, UDP retransmission, timeouts, etc.) are defined in subclasses of the superclass *Protocol*. PDU contents and format is defined in subclasses of superclass *Transaction*. Subclasses of both Protocol and Transaction are instantiated by *abstract factory*

12. Methods overriding Service.getServiceNr.

(Section 4.1.4) at the initiation of every ISMP transaction.

Currently implemented subclasses of Protocol and Transaction are shown in Figure 33. Their names are self-explanatory.

4.1.4 Abstract Factory

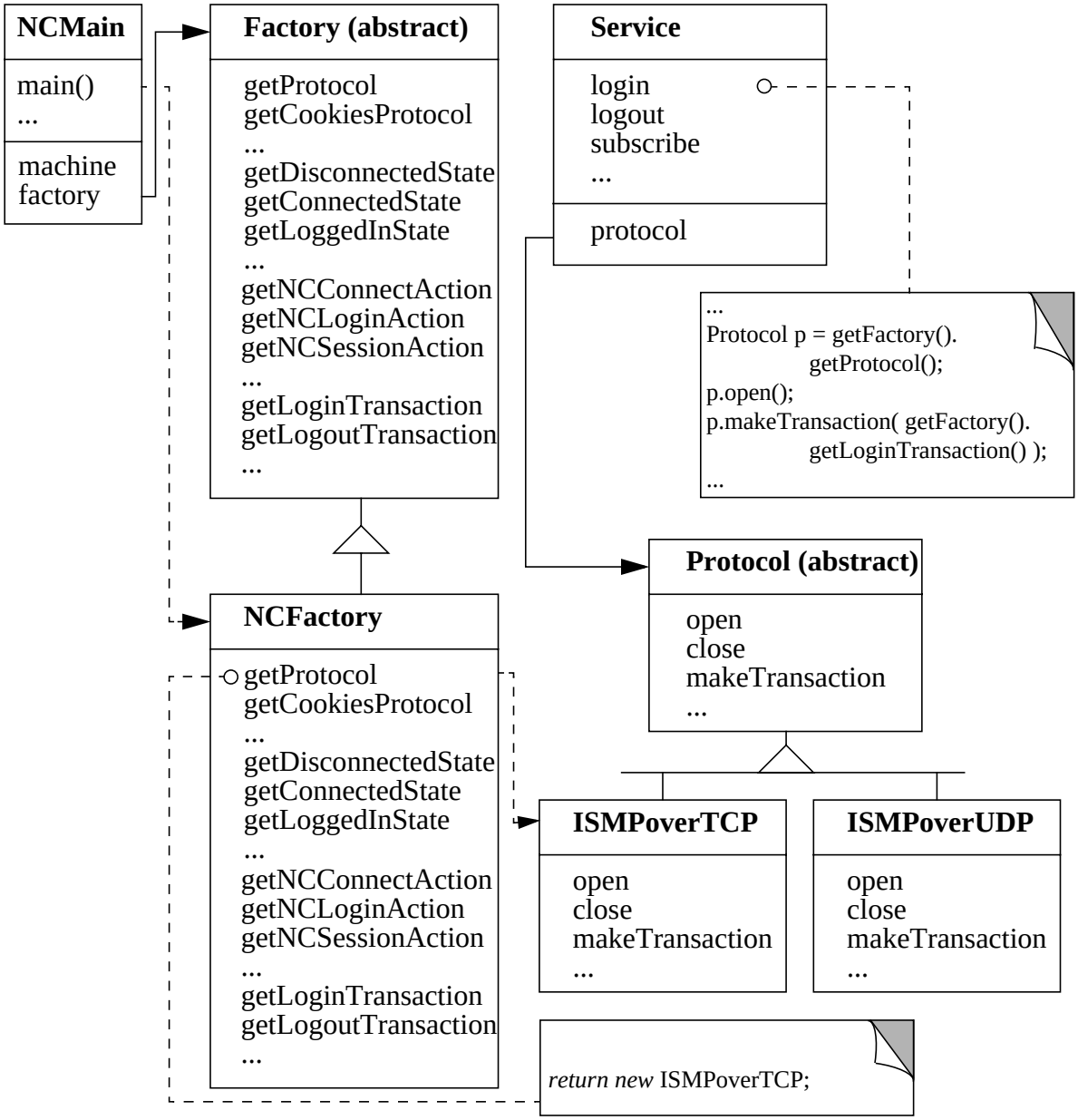

Figure 34: Factory Pattern, Example of Protocol Instantiation

Another important class pattern used is *abstract factory*. “Abstract factory provides an interface for creating families of related or dependent objects without specifying their concrete classes.” [dp]

One useful aspect of this pattern is that the NCSM behavior can be completely changed by simply exchanging the factory. The new factory may instantiate extended classes, defining a completely different ISMP format, and underlying transport protocol, SDB format, the NCSM's state machine, user interface and more. This exchange is possible even during run-time! Figure 34 depicts the class hierarchy.

4.1.5 NCSM Thread Classes

NCSM takes advantage of Java's thread support. NCSM's threads run in background and perform watching functions. The thread classes (i.e. classes derived from class *java.lang.Thread*) are listed here:

- **SubscriptionThread**. This daemonic thread waits on the SDE port (see Section 3.3 on page 57) and collects SDE requests. If a correct request comes, then service subscription is initiated by this thread.
- **CookiesThread** . This thread regularly sends "refresh cookies" (see page 22) to service servers to notify them of all logged-in services.
- **LoginTimeThread** regularly updates "Service" instances with the up-to-date current and accumulative login times.
- **SystemCommands.StreamRedirector**. This thread redirects the pipelined output of external programs started by NCSM to a file. (See Section 4.3.3.1 on page 81 for more details.)
- **One shoot message handlers**. These no-name inner classes[13] define message handling (See Section 4.2 for more details on NCSM messages.) Having one thread per message handler makes parallel message processing possible, e.g. login transactions of many services can be concurrently processed upon session initiation.

4.1.6 Integration with User Interface

The following design pattern has been used for connecting the NCSM control code with the *user interface* (UI): a *template class* defines skeleton of NCSM algorithms in an operation, and its subclasses redefine certain steps. In particular they instantiate[14] a UI object (classes *StartupDlg* and *AppServiceView* in Figure 35). The interaction between the template subclasses and UI objects utilizes the messaging system described in Section 4.2. This design pattern corresponds to well-known "*document-view architecture*" with a "one-to-one" relationship between a *document* and its *view*.

13. These inner classes are subclassed from Base.MessageHandler and defined in the classes NCConnect, NCSession and NCLogin.
14. The Base subclasses instantiate UI in the method 'startUI', the service subclasses instantiate UI in their constructors.

The design pattern is employed for classes which define the UI of the NCSM for each of its states (*Base* and its subclasses) as well as the UI of the ISMP services (class *Service* and its subclasses). See Figure 35 for more details.

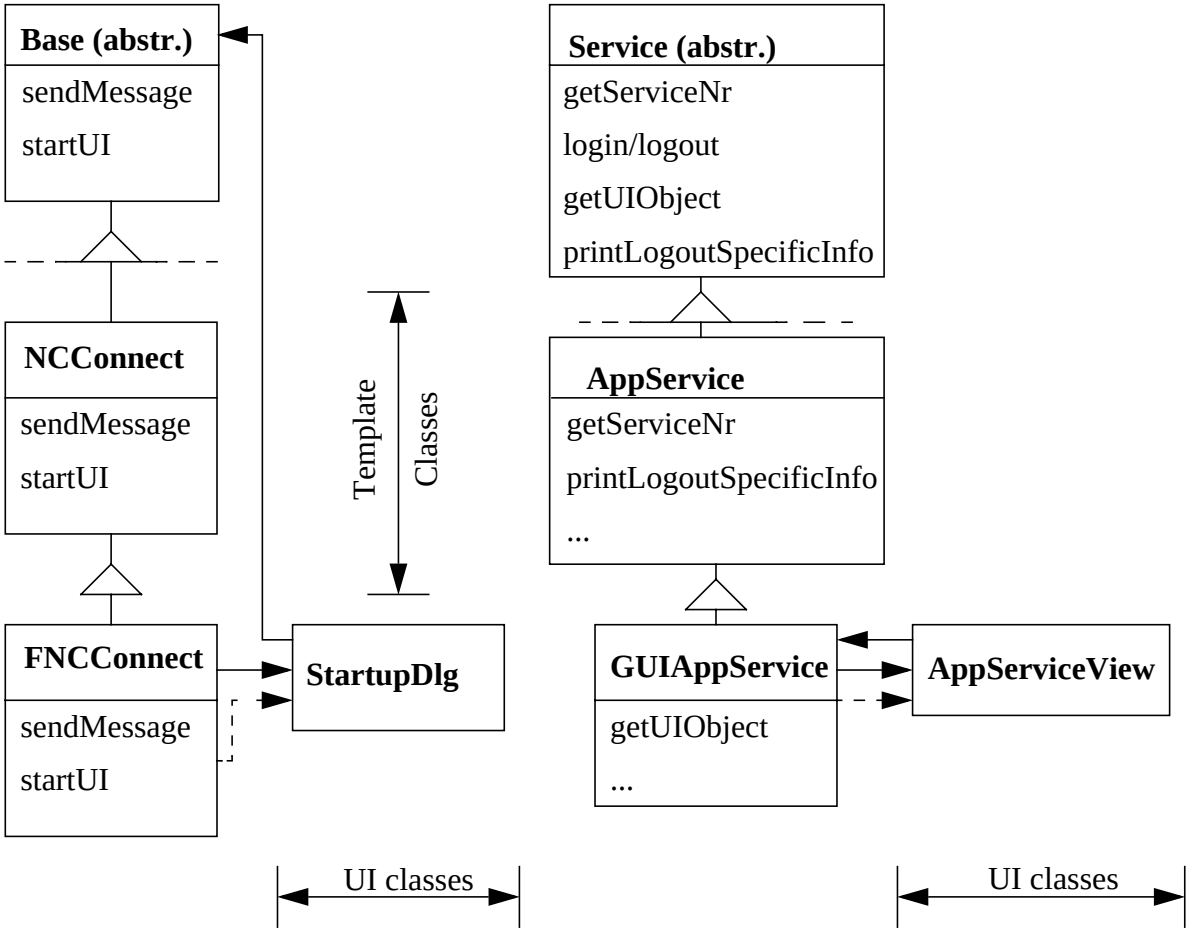

Figure 35: Class Design of UI Integration

4.2 Messaging Mechanism

The UI classes need to interact with the *Base* subclasses. They may want to change the machine's status, subscribe a service, save the SDB, terminate the NC session, etc. at the user's request. Message passing is utilized in our concept. This makes software development easier: the UI classes do not need to know which concrete subclasses of *Base* are associated with them at run-time. They just refer to an object capable of receiving messages[15] (e.g. *StartupDlg* referring to *Base* in Figure 35). Template developers can supply their message handlers, even after UI developers already have evoked them over a message. This loose coupling makes it possible to exchange the Base subclass without having any impact on the associated UI object. More-

15. The messages are sent by calling receiver's method Base.sendMessage.

over, new messages can be dynamically registered at run-time[16]. The only overhead is caused by the fact that the received messages and their parameters have to be interpreted[17].

4.2.1 Parallel Message Processing

Generally, one can distinguish between *blocking* vs. *nonblocking* messages. The latter can be classified further as either *serially* or *parallelly* processed.

The *blocking* message processing forces message *senders* to wait until message handling is completely done. This may be inconvenient because the sender may need to continue with his own processing. Therefore *nonblocking* message processing is preferred in most cases. It requires an additional mechanism to notify the sender of processing completion; see Section 4.2.2 for more details.

The next question is whether to process incoming messages one after the other (FIFO order) or concurrently. For example, the parallel transaction processing proposed in the Note on page 29 may take advantage of concurrent message handling. On the other hand, parallel processing of “login” and “logout” messages may lead to unpredictable results.

To be able to process messages parallelly and to avoid such troubles, a set of rules and constraints has been defined:

1. *Messages can be either blocking (BM) or nonblocking (NBM); the NBMs are reserved for changes of NCSM machine’s state.*
2. *NBMs may be processed concurrently and in any order.*
3. *If a BM arrives, it may not be processed until the processing of all messages currently being handled is finished; no other message may be handled while processing the BM.*
   Motivation: BMs change the NCSM’s state which causes many operations to take place accordingly. These operations might seriously interfere with NBMs currently being processed.
   Example: If there were no such constraint, a nonblocking service subscription taking a long time might be overtaken and interrupted by a blocking ICS logout. The user would have rented a service on the Internet, but no chance to put it in SDB and access it (because SDB is inaccessible and the subscription transaction is terminated after ICS logout).
   Implementation note: A stronger constraint is being implemented currently so that all messages are processed serially.[18]

16. Dynamic message registration is not currently supported.
17. Message interpretation takes place in methods which override Base.selectMessageHandler.
18. The method Base.sendMessage is synchronized, i.e. no new messages can be received until processing of a BM is finished. The method Base.MessageHandler.run is also synchronized to guarantee serial message processing. The processing order relies on the JVM executing the message handler threads in the same order in which they were launched in Base.sendMessage. A message queue will be implemented in next NCSM release to enable parallel message processing.

4. *Messages received by an invalid target (i.e. a target which is not associated with the current machine's state any more) will be ignored.*
   Motivation: The NCSM is designed as a finite state machine; according to the definition of an FSM, only the active state receives messages.
   Example: While a blocking ICS logout is being processed, a nonblocking service subscription message arrives. The NBM is blocked, according to constraint 3, until the ICS is logged out and consequently the machine's state is changed to "disconnected". It is senseless to try to subscribe to a service in this state.
   Implementation note: Such messages are simply discarded, and a warning is printed in *stderr.*[19]
5. *All ISMP operations over a service are synchronized.*
   Example: Trying to log out a service while logging the service in may lead to unpredictable results.

4.2.2 Error Handling

A nonblocking message sender does not wait until message processing finishes. In order to notify the sender of processing completion, an asynchronous notification mechanism has been established: a pair of callback methods (so called *completition handlers*) exists for each message. The first method is called when message handling succeeds, the other one when it fails. These methods are located in the receiver's class and have to be overridden if the result of the message handling is needed in order to perform further processing. A complete listing of the *completion handlers* is in Table 13 on page 76. Figure 36 shows an example of the overridden handlers.

```
/* FNCSession completion handlers of the message MSG_SERVICE_LOGIN_NB */

protected void onServiceLoginFailed(Service service, Exception e) {
        // indicate the current login status of service
        serviceView = (NCServiceView)service.getUIObject();
        serviceView.setToggleButton(service.isLoggedIn());
        // display an error message
        showAlertWindow("Service-Login", e.getMessage());
}

protected void onServiceLoginSucceeded(Service service) {
        // indicate the current login status of service
        serviceView.setToggleButton(service.isLoggedIn());
}
```

Figure 36: Example of Overridden Completion Handlers

19. Implemented in Base.sendMessage().

4.2.3 NCSM Messages

Table 13 summarizes all NCSM messages with respect to their receivers (*Base* subclasses). *Message parameters* are the names of classes from package *FOKUS.NC.messages* which encapsulate all the values passed with a message. The *Completion handlers* are the names of methods asynchronously called when message processing is finished.

Table 13: NCSM Messages

Machine State and Associated *Base* subclass	Message[a] (the suffix BL/NB stands for blocking/non-blocking message)	Message Parameters	Completion Handlers[b]	Comments
disconnected (NCConnect)	MSG_SYSTEM_CONNECT_BL	ConnectMsgPar	NCConnect.onConnectFailed onConnectSucceeded	establish ISDN connection
	MSG_SYSTEM_SHUTDOWN_BL	-	onShutdownFailed onShutdownSucceeded	shutdown NC
	MSG_PROVLIST_WRITE_NB	-	onWriteProvListFailed onWriteProvListSucceeded	serialize database of ICS providers
connected (NCLogin)	MSG_SYSTEM_ICSLOGIN_BL	-	NCLogin.onLoginFailed onLoginSucceeded	login ICS
	MSG_SYSTEM_CANCEL_BL	-	onCancelFailed onCancelSucceeded	drop ISDN connection
	MSG_ICS_SUBSCRIBE_BL	IcsMsgPar	onIcsSubscriptionFailed onIcsSubscriptionSucceeded	subscribe ICS
	MSG_OPEN_URL_NB	OpenURLMsgPar	onOpenURLFailed onOpenURLSucceeded	open a URL

Table 13: NCSM Messages

Machine State and Associated *Base* subclass	Message[a] (the suffix BL/NB stands for blocking/non-blocking message)	Message Parameters	Completion Handlers[b]	Comments
logged-in (NCSession)	MSG_SYSTEM_ICS_USUB_BL	PasswordMsgPar	NCSession.onICSUsubFailed onICSUsubSucceeded	unsubscribe ICS and its provider
	MSG_SYSTEM_ICSLOGOUT_BL	-	onICSLogoutFailed onICSLogoutSucceeded	logout ICS and drop ISDN connection
	MSG_SERVICE_LOGIN_NB	ServiceMsgPar	onServiceLoginFailed onServiceLoginSucceeded	login a service
	MSG_SERVICE_TOGGLELOCK_NB	ServicePWMsgPar	onServiceLockFailed onServiceLockSucceeded	lock/unlock a service (locked services require service password on login)
	MSG_SERVICE_LOGOUT_NB	ServiceMsgPar	onServiceLogoutFailed onServiceLogoutSucceeded	logout a service
	MSG_SERVICE_SUBSCRIBE_NB	ServicePWMsgPar	onServiceSubscribeFailed onServiceSubscribeSucceeded	subscribe a service
	MSG_PROVIDER_SUBSCRIBE_NB	ProviderSrvMsgPar	onProviderSubscribeFailed onProviderSubscribeSucceeded	subscribe a provider
	MSG_SERVICE_UNSUBSCRIBE_NB	ServicePWMsgPar	onServiceUnsubscribeFailed onServiceUnsubscribeSucceeded	unsubscribe a service

Table 13: NCSM Messages

Machine State and Associated *Base* subclass	Message[a] (the suffix BL/NB stands for blocking/non-blocking message)	Message Parameters	Completion Handlers[b]	Comments
logged-in (NCSession)	MSG_SERVICE_CHANGE_NB	ServiceChangeMsgPar	onServiceChangeFailed onServiceChangeSucceeded	change service's parameters or password
	MSG_PROVIDER_CHANGE_NB	ProviderChangePwMsgPar	onProviderChangeFailed onProviderChangeSucceeded	change identification information or password at provider
	MSG_SERVICELIST_WRITE_NB	-	onWriteConfigSucceeded onWriteConfigFailed	serialize SDB
	MSG_SERVICE_EXEC_NB	ServiceMsgPar	onServiceExecFailed onServiceExecSucceeded	log-in a service if it is logged out and launch a service browser
	MSG_OPEN_URL_NB	OpenURLMsgPar	onOpenURLFailed onOpenURLSucceeded	open a URL in a Web-browser
	MSG_PROVIDER_UNSUBSCRIBE_NB	ProviderPWMsgPar	onProvUsubFailed onProvUsubSucceeded	unsubscribe a provider

a. Defined in class NCSystemMessages.
b. Defined in their Base subclasses: NCConnect | NCLogin | NCSession.

4.3 Used Software

This subchapter describes the difficulties which may arise while using the software introduced in Section 1.3.3 on page 12.

4.3.1 Netscape Communicator

4.3.1.1 Remote Control

Netscape Communicator is used for accessing many of the services managed by ISMP (disk-like services, E-mail, news, Java applications). NCSM will control the Communicator in order to be able to explore it completely: current E-mail and news servers have to be set up at service login, the provider's homepage will be opened at the user's request, Java applets will be started, and Communicator has to be terminated gracefully at the end of every session.

Unfortunately, the command set of Netscape's built-in *remote control* [ncrc] is small and the kind of remote control used by Netscape can only be used to perform some of the necessary procedures (opening a web-page or terminating the browser, for example). Setting preferences is not possible. Therefore, either the Communicator's code has to be modified so that it can accept more commands (Netscape is going to make the source code free — see [nsfsc]) or the Netscape administration tool *Mission Control* [nsmc] has to be employed.

4.3.1.2 Miscellaneous

- The Communicator's cache must be located at a local directory in order to prevent storing the cache in the remote NFS-exported ICS-directory.[20]
- The cache has to be deleted after every session for privacy reasons.[21] (Consider a curios person examining contents of your cache at a public NC after you have logged out.)
- The large Communicator's binary will be read into the NC's memory in advance in order at accelerate its launch on ICS-login.[22]

4.3.2 Linux Tools

There were several difficulties while using the underlying utilities distributed with Linux (distribution *Slackware 3.1*). They are listed in this section.

20. The creation of the cache directory is implemented in the script $BROWSERWRAPPER (which is dynamically created in script init-nc). The setup of the directory in Netscape preferences is implemented in the script ics-sub.
21. Implemented in scripts ics-off and ics-on.
22. Implemented in X start-up script xinitrc.nc.

4.3.2.1 Starting X

According to the defined process structure (Figure 25 on page 55), the *init* process starts *xinit*, the X Window System initializer. Xinit is responsible for starting the *X server* and a shell-script[23] from which *NCSM* is started. If either *X server* or the shell-script terminates, or *xinit* receives *HUP* signal, *xinit* terminates the X session.

The problem is *init* sends the *TERM* signal while changing the run-level, and this signal is *NOT* handled by *xinit*. If *init*'s run-level is changed, *xinit* ignores the *TERM* signal and, after a certain period of time, it is mercilessly killed by *init*'s *KILL* signal. The *X server* as well as the NCSM keep running, because the *xinit* has no chance to kill them.

This problem has been fixed by simply handling the *TERM* signal in the same way as the *HUP* signal is handled.

4.3.2.2 Chroot and Su Commands

Chroot and *su* commands are utilities which are used by NCSM to create an invulnerable environment (as is required in Section 3.2 on page 47). *Chroot* limits all the file operations of a process to a part of the filesystem while protecting the rest against unauthorized access. *Su* changes the effective uid of the superuser under which the NCSM is run to the uid of the NC user, in order to prevent him or her from accessing the NC resources with unlimited privileges. This arrangement greatly reduces the impact of possible security loopholes in service management software. (The *chrootid* utility [chr] might be used instead.) Unfortunately, neither *chroot* nor *su* is perfect.

Linux *chroot* does *not* change the *working directory*. If the current working directory is located beyond the chroot-ed directory, the chroot-ed process can still access it!

GNU *su*, distributed with Linux, allows users to specify a start-up shell with the command line option "-s <shell>". This feature may be misused to gain superuser privileges! Let us look at an example: Some UNIX administrators decide to have a special user account with no password and which has a rebooting sequence as a login shell. This allows any user without special privileges to shut-down a workstation. But if the users run *su* in order to become the "shutdown user" and specify Bourn-shell as login shell (instead of the shell specified in /etc/passwd), they gain privileged access to the workstation.

In our project, the *chroot* utility has been fixed and a version of *su* which does not support the shell option has been used. The *chrootid* will be used in next implementation versions.

23. xinitrc.nc

4.3.3 JDK 1.1.3 for Linux

4.3.3.1 Execution of Subprocesses

The NCSM starts many subprocesses, especially because all of the OS-related functions are performed by Bourne-shell scripts (see Section 3.2.1). Not only do subprocesses have to be started (which is easy[24]), but also their output streams have to be redirected and their terminations have to be watched in order to find out *when* and *how* (i.e. with what termination code) they terminated.

To understand why subprocesses' output streams (i.e. standard output and standard error) have to be redirected, Linux JVM's subprocess handling has to be well understood: When JVM starts a subprocess, it creates three *pipes* for standard input, output and error. The file-descriptors of these pipes are passed to the subprocess. They can be accessed over Java API[25]. A problem arises if you wait for the termination of the child process but do not read in its output (even if you do not need it). The child process pushes its output into the pipe buffer and gets blocked as soon as the buffer is full, while you are waiting for its terminations. This is a *deadlock* situation which has to be prevented by reading-in and redirecting the output of the subprocess[26].

Asynchronous awaiting subprocesses' termination is not possible due to Java's lack of signal handling. But there exists a trick which allows one to await the termination while processing other procedures: a new thread is started for each child process. The thread does nothing but wait for a child's termination. Afterwards, it handles the termination as a signal handler would.

Another problem is that various JDK ports return different *termination codes* of started child processes. Linux JDK returns the status returned by a 'wait' system call (a combination of child's exit value and a signal number), Solaris JDK returns either the child's exit value or a signal number without specifying which of these two values has been actually returned. The only OS-independent unambiguous termination code is zero, which states, 'the child regularly terminated with exit code 0'. Examples are given in Table 14.

Table 14: Examples of JDK Return Codes, Linux vs. Solaris

Type of Child's Termination	JDK Exit Value		Comment
	Linux	Solaris	
exit 0	0	0	the only OS-independent unambiguous termination code

24. java.lang.Runtime.exec
25. java.lang.Process.{getErrorStream | getOutputStream | getInputStream}
26. Implemented in SystemCommands.StreamRedirector.

Table 14: Examples of JDK Return Codes, Linux vs. Solaris

Type of Child's Termination	JDK Exit Value		Comment
	Linux	Solaris	
exit 1	256	1	Solaris JDK does not distinguish between termination of a subprocess by a signal and a regular termination by call to 'exit'. Linux JDK passes the value returned by the 'wait' system call (i.e. child's exit code in the high order 8 bits, termination signal number in the low order 8 bits).
kill 1	1	1	

4.3.3.2 Socket Timeout Handling

JDK functions are mapped to different system calls depending on which operating system is being used. For example, the Linux JDK implementation of handling socket timeouts relies on the system call 'select' (unlike Solaris JDK). This system call is interruptible, and JVM throws an exception[27] if the call is actually interrupted by a signal[28]. Therefore, the exception will be thrown[29] in Java programs which enable socket timeouts and are ported to Linux.

4.4 NCSM Reference

This chapter provides a tabular overview of the NCSM distribution.

The NC Session Manager (NCSM) is downloaded to the NC upon OS-update (see Section 3.4 on page 61) along with the underlying OS and all of the utilities required by the NCSM (Table 17). The NCSM package (Table 16) has to be the last to be installed, because it overrides some default system settings and utilities. After a successful OS-update, the NC is rebooted at run-level 7. A sequence of start-up scripts[30] is started by *init* process to launch the X Window System and the NCSM with options summarized in Table 15. The resulting process structure has been introduced in Figure 25 on page 55.

27. SocketException "interrupted system call" is thrown
28. In our 'trace' observations the 'select' call was always interrupted by an 'ALRM' signal and returned an 'ERESTARTNOHAND' error value
29. Not implemented yet.
30. rc.nc, rc.nc.local, rc.nc.d; the *respawn* option is set in *inittab.*

Table 15: NCSM Command Line Options

Syntax	Meaning	Status
-sernr <NC serial nr.>	serial number of NC; this unique device-specific number is transmitted with every ISMP request; see Section 2.4.1 on page 32	mandatory
-ncdir <NC chroot-ed directory>	absolute path of the directory where user subfile-system is set-up (see Section 3.2.3 on page 53)	mandatory
-sysdir <NC system dir>	absolute path of the directory where NCSM management is stored[a]	mandatory
-ncinfo <NC information>	additional information on the NC's software and hardware transmitted with every ISMP request; see Section 2.4.1 on page 32	mandatory
-browser <browser path>	absolute path of Web-browser	mandatory
-help	print a brief summary of the allowed options and exit	optional
-dbg	verbose mode (debugging feature)	optional
-udp	run ISMP over UDP instead of TCP	optional
-port <port number>	the number of the ISMP port; default value is 9000[b]	optional
-server <hostname>	the hostname of the local login server; default value is *"logins"*[c]	optional
-dbgfactory	enable "DebugFactory" instead of default "NCFactory" (debugging feature); see Section 4.1.4 for more details on *Abstract Factory*	optional
-configfile <filename>	the absolute path to SDB; default value is "/usr/nc/home/.nc/services.nc"[d]	optional
-nosub	turn off SDE server (debugging feature)	optional
-suburl <ICS subscription URL>	URL of ICS subscription Web-page (see Section 3.3.2); default value is "http://www/ics-subs.html"[e]	optional
-timeout <milisec.>	ISMP response timeout (see Table 3 on page 22); the default value is 3 minutes[f]; setting this value to zero disables the timeout	optional

a. /opt/FOKUSnc currently.

b. Defined in Protocol.PORT_NR.
c. Defined in NCMain.ICSServer.
d. Defined in NCSession.SERVICESFILE.
e. Defined in NCMain.subscriptionURL.
f. Defined in Protocol.TC_TIMER_RES.

Table 16: Filesystem Structure of NCSM Distribution

File/Directory	Description
ncos	symbolic link to OS-kernel
ncos-<version>	OS-kernel
/bin	updated system utilities
/etc/exports	permission to export filesystems to the local host over NFS
/etc/inittab	definition of NCSM's run-level
/etc/ld.so.conf	list of NCSM's shared libraries
/etc/rc.d/rc.nc	NCSM's run-level start-up script
/etc/rc.d/rc.nc.local	device-specific NCSM's run-level start-up script; called by rc.nc; currently, display resolutions is set-up in this script
/opt/FOKUSnc/bin	NCSM utilities (patched chroot, spgrp, etc.)
/opt/FOKUSnc/etc/fstab	list of filesystems mounted on NCSM's start-up by the 'mountlocal' script
/opt/FOKUSnc/etc/rc.nc.d	continuation of the rc.nc script
/opt/FOKUSnc/etc/skel	templates for "/etc/" files created in the user's subfilesystem upon NCSM's initialization in the script *init-nc*; all symlinked files in this directory are copied to /usr/nc/etc
/opt/FOKUSnc/etc/usrnc.tar	the archived initial contents of user's subfilesystem; directory structure and device files are included; the package is expanded in the script rc.nc.d
/opt/FOKUSnc/etc/xinitrc.nc	X Window start-up script; NCSM is started from this script
/opt/FOKUSnc/lib/classes	NCSM Java code
/opt/FOKUSnc/lib/netscape*	Netscape IFC Classes (used for NCSM GUI)
/opt/FOKUSnc/sbin	NCSM OS-related scripts (see Section 3.2.1 for more details; all the scripts are listed in Appendix B)

Table 16: Filesystem Structure of NCSM Distribution

File/Directory	Description
/usr/local/jre/bin	patched JRE start-up script (the original script is not correctly ported to Linux)

Table 17: List of Linux Tools Required by NCSM

Name	Description
awk	pattern scanning and processing language
bash	GNU Bourne-Again Shell
bc	arbitrary precision arithmetic language
cat	concatenate and display files
chmod	change the permissions mode of a file
chroot	change root directory for a command; patched, see Section 4.3.2.2 for details
dirname	deliver portions of path names
dc	desk calculator
echo	echo arguments
egrep	search a file for a pattern using full regular expressions
find	find files
fuser	identify processes using a file or file structure
head	display first few lines of files
isdnbutton	a simple ISDN monitor
isdnctrl	get/set ISDN device information
java	The Java Interpreter
inorootfs	test a file if it is located in the root filesystem or not
killall	kill processes by name
ldconfig	determine run-time link bindings
ln	make hard or symbolic links to files
mkdir	make directories
mount	mount filesystems and remote resources; patched to support NFS uid mapping (see Section 3.1.5 for more details)
named	Internet domain name server

Table 17: List of Linux Tools Required by NCSM

Name	Description
netscape	Netscape Communicator
nslookup	query name servers interactively
ps	report process status
rdate	set system date from a remote host
rm, rmdir	remove directory entries
route	show / manipulate the IP routing table
sed	stream editor
sleep	suspend execution for an interval
spgrp	establish a new process group and record its id (see Section 3.2.3, Footnote 13 for more details)
su	become super-user or another user
tar	archiving utility
toolwait	control client program start-up (OpenWin program)
umount	unmount filesystems and remote resources
WM	a window manager
xinit	X Window System initializer; patched (see Section 4.3.2.1 for more details)
xrdb	X server resource database utility
xsetroot	root window parameter setting utility for X
X	X server

4.5 Implementation Tests

The implemented Java code has been extensively tested to verify the theoretical predictions for ISMP reliability and performance. The tests proved that *all* of the assumptions made in Section 2 are valid:

- Both *usage* and *charging hazards* occurred rarely due to the built-in reliability features — especially *refresh cookies* (page 22) and predefined *hazard recovery* (Table 4 on page 24). The charging hazard was always successfully converted into a less harmful usage hazard. No other inconsistencies were recognized during the tests.

- The redundant “Confirmation of requested service specific parameters” ISMP fields (Figure 12 on page 33) turned out to be very helpful after the SDB had been damaged.
- The ISMP traffic has been observed at the IP level. The number of IP-packets exchanged within one transaction (see the section entitled “One ISMP Transaction over One TCP Connection (OoO)” on page 25) as well as transaction times over ISDN were in agreement with the theoretical estimations presented in Section 2.3.2.4 on page 27. (The experimental values were obtained with the *snoop* utility, see Figure 37.) This result justifies the selection of the “1 over 1” transaction mechanism made in the Section 2.3.2.4 on page 27.

Table 18: Summary of ISMP Observations on IP Level (application-login transaction)

measured quantity	obtained value	comment
number of IP packets	13	According to presumptions, it should be 11. ISMP implementations sends the PDU in two parts which increases the number of IP packets.
Round Trip Time [ms]	29	This is the delta-time of packet #3 in Figure 37. Typical value for ISDN connection.
Server Processing Time [ms]	866	This is the delta time of packet #8 in Figure 37. (This is an application-service specific value. Note, that this is an approximate value, because all debugging features of the experimental server are enabled and the server is started by ‘inetd’.)
Client Processing Time (CPT)	N/A	CPT can not be read from Figure 37 because the processing takes place after network operations. This value ranges in tenths of seconds according to local measurements. (see also Section 3.2.1.1 on page 48)
Total processing time [s]	ca. 1.5	$(CPT \approx x \bullet 100) + \left(\frac{1}{2}RTT \approx 15ms\right) + \sum_{i \in \langle 2;11 \rangle} \Delta_i \approx$ circa 1.5 s

Note: The transaction time for services provided over the Internet has not been measured. The Internet only guarantees an indeterministic “best-effort” quality of service. Measurement results would strongly vary with the current Internet load and route to the service provider.

Although the ISMP’s correctness has not been formally verified (see the section entitled “Formal Protocol Specification and Verification” in the bibliography for references), practical experiments have demonstrated very satisfactory performance and error recovery.

```
/* 'nc' is the Network Computer, "logins" is the Login Server; the timestamps (second column) are presented
in delta format (the time since receiving the previous packet) */

/* TCP connection establishment */
1   0.00000   nc -> logins    D=9000 S=1211   Syn Seq=207521028 Len=0 Win=512
2   0.00017   logins -> nc    D=1211 S=9000   Syn Ack=207521029 Seq=1734142511 Len=0 Win=8760
3   0.02869   nc -> logins    D=9000 S=1211   Ack=1734142512 Seq=207521029 Len=0 Win=31744

/* ISMP request -part 1 */
4   0.00721   nc -> logins    D=9000 S=1211   Ack=1734142512 Seq=207521029 Len=8 Win=31744
5   0.00022   logins -> nc    D=1211 S=9000   Ack=207521037 Seq=1734142512 Len=0 Win=8760
/* ISMP request -part 2*/
6   0.03051   nc -> logins    D=9000 S=1211   Ack=1734142512 Seq=207521037 Len=82 Win=31744
7   0.09326   logins -> nc    D=1211 S=9000   Ack=207521119 Seq=1734142512 Len=0 Win=8760

/* ISMP response */
8   0.86624   logins -> nc    D=1211 S=9000   Ack=207521119 Seq=1734142512 Len=253 Win=8760
9   0.07049   nc -> logins    D=9000 S=1211   Ack=1734142765 Seq=207521119 Len=0 Win=31744

/* TCP connection released by client */
10  0.11996   nc -> logins    D=9000 S=1211   Fin Ack=1734142765 Seq=207521119 Len=0 Win=31744
11  0.00005   logins -> nc    D=1211 S=9000   Ack=207521120 Seq=1734142765 Len=0 Win=8760
12  0.00268   logins -> nc    D=1211 S=9000   Fin Ack=207521120 Seq=1734142765 Len=0 Win=8760
13  0.02236   nc -> logins    D=9000 S=1211   Ack=1734142766 Seq=207521120 Len=0 Win=31744
```

Figure 37: ISMP Transaction (application login) Captured on IP-level with Snoop

5 Summary

5.1 Summary (US)

This paper has introduced the extension of today's challenging Network computing architecture proposed by FOKUS. The benefits of this extension are higher mobility, the allocation of more resources on the net, electronic service subscription and minimized administration and maintenance overhead.

The central means for support of these features, the TCP/IP based Internet Service Management Protocol (ISMP), has been defined, discussed and analyzed in Chapter 2. Generally, this protocol automatizes the subscription and usage of services provided over the Internet. Special attention has been paid to reliability of the ISMP in guaranteeing error recovery and fair charging for services even in the case of a failure.

Chapter 3 discussed the following: technical solutions to pitfalls resulting from the innovative concept, various aspects of the service management application (NCSM), means for electronic service subscription and issues which shall be addressed in the future.

The extensive tests of protypical Java implementation described in Chapter 4 provide evidence for the technical functionality and reliability of this extended architecture.

5.2 Zusammenfassung (D)

In dieser Diplomarbeit wird die beim Forschungsinstitut 'FOKUS' entworfene Erweiterung der heutigen "Network computing" Architektur vorgestellt. Die Vorteile dieser Erweiterung sind insbesondere höhere Mobilität, Lokalisierung mehrerer Ressourcen im Netzwerk, elektronische Dienstbestellung und minimaler Verwaltungs- und Wartungsaufwand.

Das TCP/IP basierte "Internet Service Management Protocol" (ISMP) ist zur Unterstützung dieses Ansatzes entworfen und analysiert worden. Eine ausführliche Spezifikation ist im Kapitel 2 zu finden. Im allgemeinen automatisiert dieses Protokoll Bestellung und Benutzung von Diensten, die über das Internet angeboten werden. Insbesondere war dabei die Zuverlässigkeit des Protokolls berücksichtigt, damit Fehlerbehebung und exakte Abrechnung auch in fehlerhaften Situationen gewährleistet werden kann.

Das Kapitel 3 beschäftigt sich mit technischen Lösungen der durch das innovative Konzept entstandenen Problemen, verschiedensten Aspekten der Dienstverwaltungssoftware (NCSM), Ansätzen zur Unterstützung automatischer Dienstbestellung und schließlich mit Problemen, die erst in der Zukunft endgültig gelöst werden werden.

Die Tests der im Kapitel 4 beschriebenen, prototypischen Java-Implementierung sind ein Nachweis der Einsatzfähigkeit und Zuverlässigkeit dieser erweiterten Architektur.

5.3 Shrnuti (CZ)

V teto diplomove praci je predstaveno rozsireni sitove architektury zvane “Network computing”. Vyhodami tohoto rozsireni jsou zejmena vyssi mobilita, alokovani vice systemovych zdroju na siti, elektronicke objednavani sluzeb, minimalni naklady na spravu a udrzbu systemu.

Zakladnim prostredkem teto rozsirene architektury je “Internet Service Management Protocol” (ISMP), protokol zalozeny na bazi TCP/IP a urceny k automatizaci objednavani a pouzivani sluzeb nabizenych pres celosvetovou sit Internet. Obzvlastni pozornost byla venovana spolehlivosti protokolu, aby mohla byt zarucena jeho schopnost zotavit se z kritickych situaci a za kazdych okolnosti poskytovat korektni uctovani pouzivanych sluzeb. Protokol je podrobne popsan a analyzovan v kapitole 2.

V kapitole 3 jsou popsana pouzita technicka reseni a problemy vznikle pouzitim inovativniho konceptu, nejruznejsi aspekty ridici aplikace pro spravu sluzeb (NCSM) a prostredky pro elektronicke objednavani sluzeb. Dale jsou nastineny problemy, ktere budou reseny az v dalsich etapach vyvoje teto architektury.

Testy prototypu implementovanem v jazyce Java a popsanem v kapitole 4 dostatecne prokazaly pouzitelnost a spolehlivost celeho konceptu.

A. Screendumps of NCSM User Interface

Figure 38: The “Connect “ Dialog

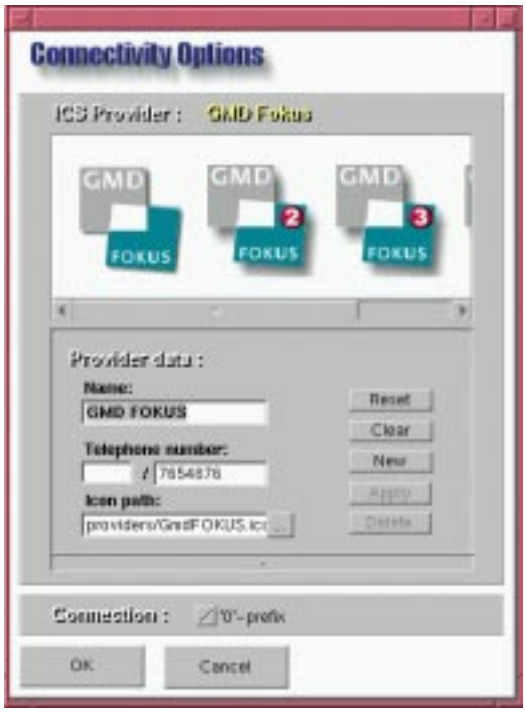

Figure 39: The “Connectivity Options” Dialog

Figure 40: The “Login” Dialog

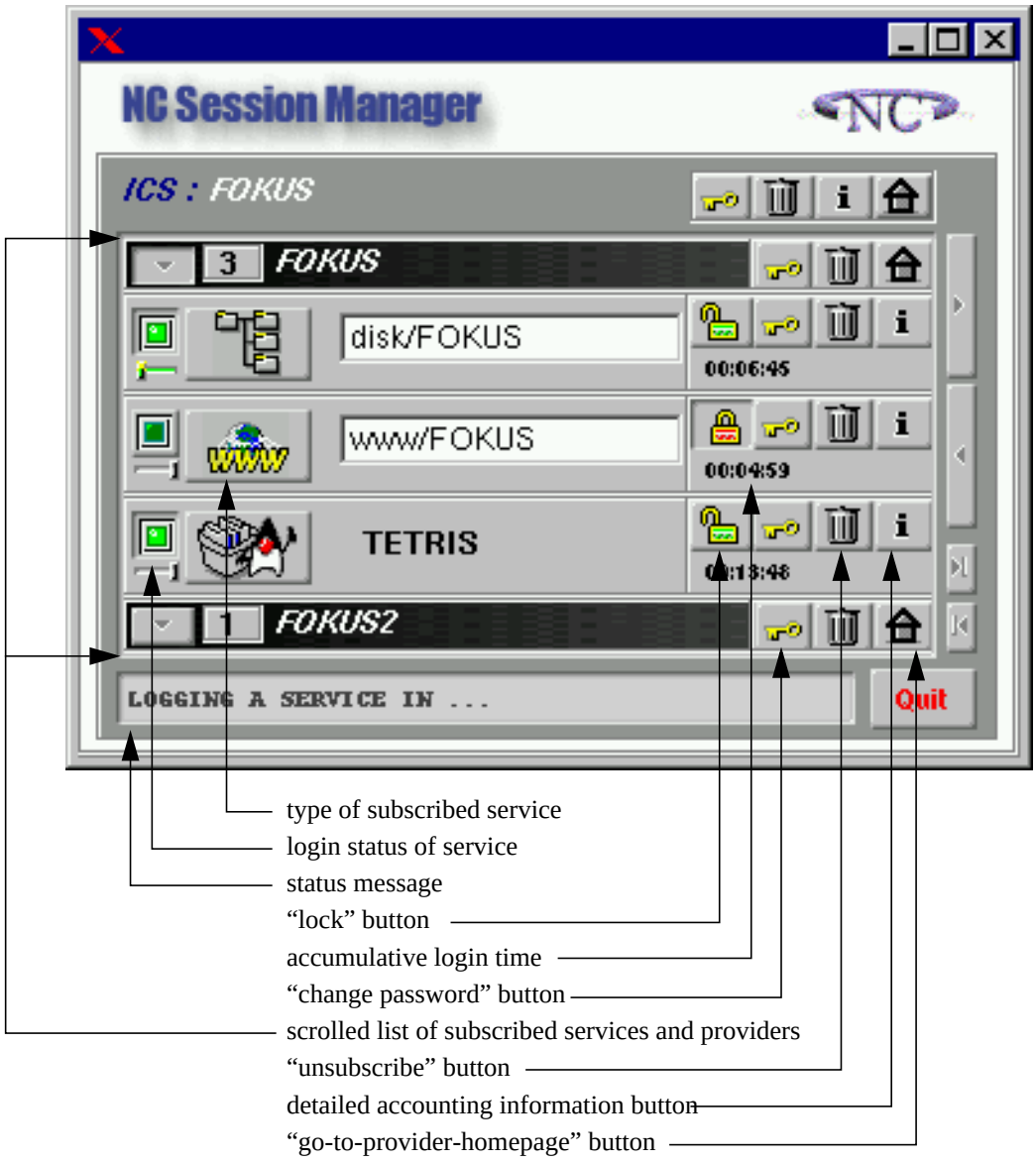

Figure 41: The NCSM Control Panel

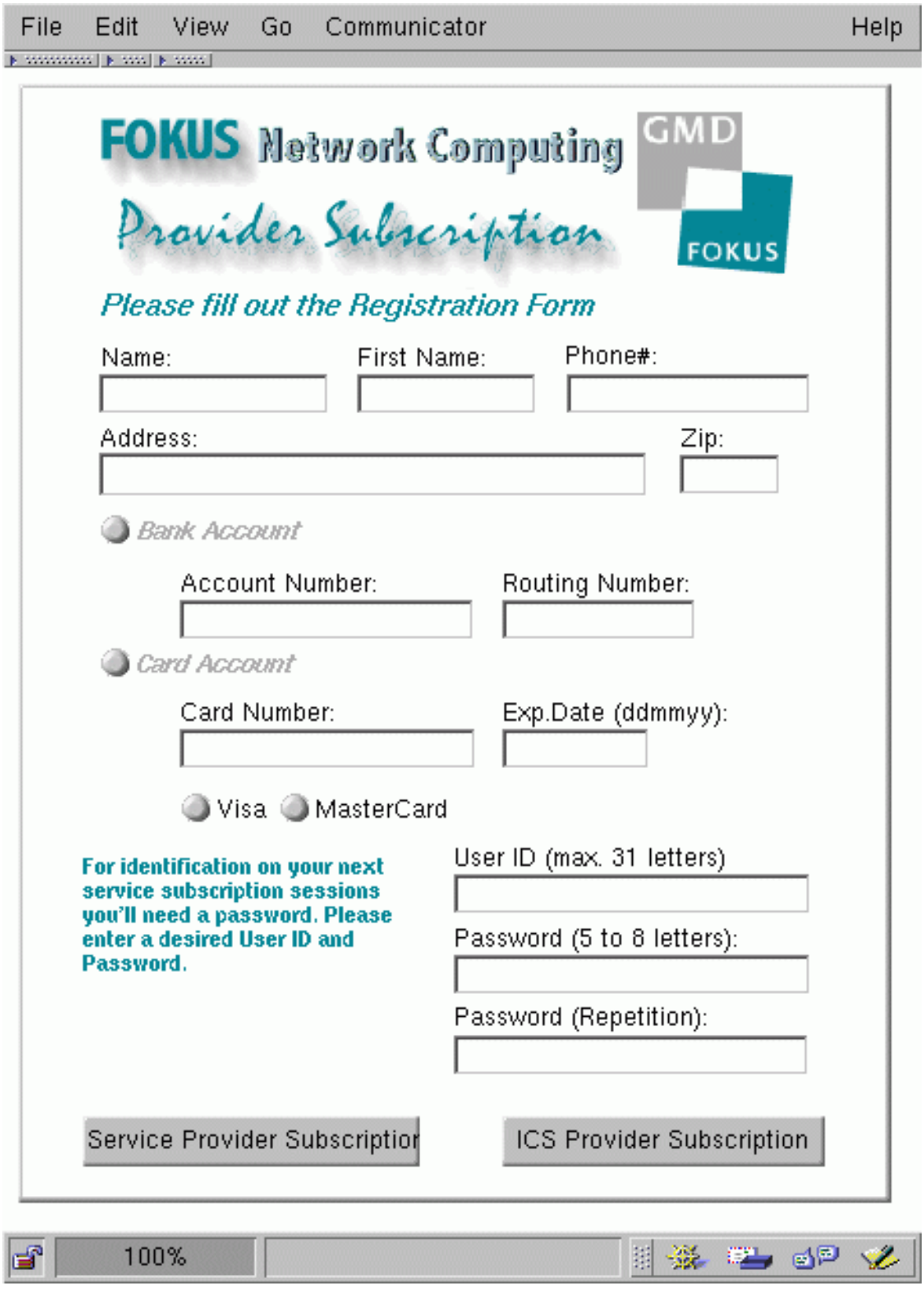

Figure 42: Provider-customized ICS-subscription Web-applet

Appendix A. Screendumps of NCSM User Interface

B. Catalog of NCSM Scripts

Table 19: Catalog of NCSM Scripts

script-name	• parameters	time-to-call	functionality
appl-exec	• spgrp filename • application URL	application-service-execution button pressed in NCSM	opening a new Netscape window with a specified URL using the Netscape remote control
appl-off	none	upon logging out from an application service	none
appl-on	none	upon logging in to an application service	none
connect	• dial-number	at the session's beginning	ISDN connection establishment
disconnect	none	at the session's end	dropping the ISDN connection
disk-exec, ftp-exec, www-exec	• spgrp filename • user id • local mount point	disk-service-execution button pressed in NCSM	starting a file-manager in a chroot-ed environment
disk-kill, ftp-kill, www-kill	• mounted directory • spgrp filename	upon logging out from a disk service	• killing all processes tagged in the spgrp-file • killing processes accessing the mounted disk with fuser
disk-off, ftp-off, www-off	• mounted directory	upon logging out from a disk-service	• unmounting a remote disk • removing a local mount-point

Table 19: Catalog of NCSM Scripts

script-name	• parameters	time-to-call	functionality
disk-on, ftp-on, www-on	• user name • remote user id • remote group id • user full name • remote server • remote directory • mount point • local user id • local group id	upon logging out from a disk service	• setting up an NFS user-id mapping • mounting a remote directory
fstabset	none	after a successful OS-update	swapping active and backup partitions in /etc/fstab
ics-exec	• spgrp filename • user name	after successful logging in to the ICS	starting the Netscape and a window manager in a chroot-ed environment
ics-kill	• local mount-point • spgrp filename	before logging out from the ICS	• killing all processes tagged in the spgrp-file • killing processes accessing the mounted disk with fuser
ics-off	• local-mount point	before logging out from the ICS	unmounting a remote disk

Table 19: Catalog of NCSM Scripts

script-name	• parameters	time-to-call	functionality
ics-on	• user name • remote user id • remote group id • user full name • remote server • remote directory • mount point • rdate server	upon logging in to the ICS	• creating local passwd entry • setting local time • setting up an NFS user-id mapping • mounting remote directory
ics-sub	• user name • provider's URL • user's home directory	after a successful ICS subscription	creating initial user settings (especially Netscape preferences)
init-nc	none	upon NCSM-start	chroot-ed environment is dynamically created
liloset	none	after a successful OS-update	LILO is set up to boot the new OS version
ncosupd	• remote filesystem • version of new OS • local OS-version filename	during an OS-update	• remote OS filesystem is mounted • new OS version is downloaded and installed • remote OS filesystem is unmounted • liloset and fstabset is called
start-sub	• Netscape command-line parameters	before ICS-subscription	• creating chroot-ed environment • starting Netscape

C. Glossary of Abbreviations

3WHS	3-way handshake protocol; TCP, ISMP
ACK	Acknowledgment flag, TCP header
ANSI	American National Standards Institute
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange; character set established by ANSI
ATM	Asynchronous Transfer Mode
B-ISDN	Broad-band ISDN
BOOTP	Bootstrap protocol [RFC951]
CHAP	Challenge-Handshake Authentication Protocol [rfc1334]
CR	Carriage Return (ASCII character)
DDE	Dynamic Data Exchange; Microsoft Interprocess Communication
DHCP	Dynamic Host Configuration Protocol [rfc1531]
DNS	Domain Name Service [rfc1034, rfc1035]
DSL	Digital Subscriber Line; challenging modem technology
EOL	End of Line (either LF or CR-LF)
FIFO	First In First Out; also “named pipes”
FIN	finish flag; TCP header
FSM	finite state machine
FTP	File Transfer Protocol [rfc765]
GNU	GNU is Not UNIX; http://www.gnu.ai.mit.edu/
GUI	Graphical User Interface
HTML	HyperText Mark-up Language [rfc1866]
HTTP	HyperText Transfer Protocol [rfc2068]
IAP	Internet Access Point
IFC	Internet Foundation Classes; Netscape class library
ICS	Internet Connectivity Service (Table 2 on page 18)
ICS HS	ICS Homeserver
IEEE	The Institute of Electrical and Electronics Engineerings
ISDN	Integrated Services Digital Network
ISMP	Internet Service Management Protocol (Section 2 on page 15)
IP	Internet Protocol [rfc791]

IPCP	Internet Protocol Control Protocol; PPP [rfc1332]
JDK	Java Developer’s Kit
JVM	Java Virtual Machine
JRE	Java Runtime Environment
LAN	Local Area Network
LF	Line Feed
LS	login server (Section 1.3.2 on page 10)
MSS	Maximum Segment Size; TCP
NC	Network Computer (Section 1.3.2 on page 10)
NFS	Network File System [rfc1094, rfc1813]
NIS	Network Information System
N-ISDN	Narrow-band ISDN
NCSM	NC Session Manager, standalone Java application (Section 1.3.2 on page 10, Section 3.2 on page 47)
OMT	Object Modelling Technique [RBP+91, Rum94]
OS	Operating Systems
PAP	Password Authentication Protocol [rfc1334]; PPP
PDU	Protocol Data Unit
POSIX	Portable Operating System Interface; IEEE standard
PPP	Point-to-point protocol [rfc1661]
QoS	Quality of Service
RFC	Request for Comment
RL	Run-level (init process)
RPC	Remote Procedure Call
RTT	Round Trip Time
SDB	Service Database (Section 3.1.6 on page 46)
SDE	Subscription Data Exchange (Section 3.3 on page 57)
SDL	Specification and Description Language
SMTP	Simple Mail Transfer Protocol [rfc821]
SFTP	Simple FTP [rfc913]
SSL	Secure Socket Layer [ssl]
SYN	Synchronize sequence numbers flag; TCP header
TCP	Transmission Control Protocol [rfc793]
T-FTP	Trivial FTP [rfc783]

T/TCP	TCP for Transactions [rfc1379, rfc1644]
URL	Uniform Resource Locator [rfc1738]
WAN	Wide Area Network
WM	Windows Manager
WWW	World Wide Web [rfc1866, rfc 2068]
XDR	External Data Representation Standard [rfc1014]

D. Bibliography

Economical Issues

[gg] Gartner Group: Management Strategies to Control the Rapidly Escalating Costs of Distributed Computing [Stanford, CT: Gartner Group 1995]

Formal Protocol Specification and Verification

[ekp] Ina Schieferdecker: Entwurf von Kommunikationsprotokollen
http://www.fokus.gmd.de:80/step/employees/ina/courses/lv3.html

[sdlfs] SDL Forum Society: Official Homepage
http://www.sdl-forum.org/

[sdlhp] Tele Danmark R & D: Specification and Description Language
http://www.tdr.dk/public/SDL/

[sdltl] Telelogic: SDL - Formal Object-oriented Language for Communication Systems
http://www.telelogic.com/sdl/sdlmain.htm

Internetworking

[ComI-III] Douglas E. Comer: Internetworking with TCP/IP, Vol. I-III [Prentice Hall]

[SteI-III] W. Richard Stevens: TCP/IP Illustrated [Addison Wesley]

ISDN

[kk] Kahnbach, A; Körber, A: ISDN: Die Technik [Hüthig Verlag]

[isdn] Börwald, W.: Kommunikationsnetze und -dienste, ISDN [VMS Verlag Modernes Studieren]

[isdnug] Ascend: ISDN User's Guide
http://www.ascend.com/1097.html

Java

[core] Gary Cornell: core JAVA [SunSoft Press - A Prentice Hall Title]

[jdk] Sun Microsystems, Inc.: JDK 1.1 Documentation
http://www.javasoft.com:80/products/jdk/1.1/docs/index.html

[jin] David Flanagan: Java in a Nutshell [O'Reilly & Associates, Inc.]

[jls] Sun Microsystems, Inc.: Java Language Specification
http://www.javasoft.com/doc/language_specification/index.html

[jlwp] Sun Microsystems, Inc.: The Java Language: An Overview
http://java.sun.com/docs/overviews/java/java-overview-1.html

Multimedia Services

[cqosl] Jan de Meer: Collection of QoS Links
http://www.fokus.gmd.de/step/employees/jdm/jdmQoSIG.html

[qosna] “QoS Negotiation and Adaption” Project, [University of Montreal]
http://www.iro.umontreal.ca/labs/teleinfo/CITR.html

[qosnd] A. Hafid, G. Bochmann: “Quality of Service Negotiation in News-on-Demand Systems, [Proceedings of the 3rd International Workshop on Protocols for Multimedia Systems, 1996, Madrid]

[rsvp] Steven Bronson: RSVP Reservation Protocol
http://www.isi.edu/div7/rsvp/rsvp.html

[rtp] Henning Schulzrinne: About RTP
http://www.cs.columbia.edu/~hgs/rtp/

Netscape Communicator

[nsfsc] Netscape: Free Source Code Press Release
http://home.netscape.com/newsref/pr/newsrelease558.html

[nsmc] Netscape: Mission Control
http://home.netscape.com/comprod/products/communicator/mission_control/overview.html

[nsrc] Jamie Zawinsky, Netscape: Remote Control of UNIX Netscape
http://home.netscape.com/newsref/std/x-remote.html

Network Computing

[knc] Network Computer, Inc.: The Key to Network Computing
http://www.nc.com/

[nca] Oracle Corporation: Network Computing Architecture
http://www.oracle.com/nca/html/nca_wp.html

[ncom] Lutz Henckel, Jörg Schilling, Thomas Baumgart, Jiri Kuthan: Network Computing in einem offenem DV-Dienstemarkt [Proceedings for the Workshop on Java in Telecommunications, Darmstadt ‘97]

[ncosm] Lutz Henckel, Jiri Kuthan: The Network Computer for an Open Service Market [Proceedings for the IFIP Conference on High Performance Networking, Vienna ‘98]

[ncrp] Apple, IBM, Oracle, Sun, Netscape: NC Reference Profile
http://www.nc.ihost.com

[ncwp] Sun Microsystems: Network Computing Whitepapers
http://www.sun.com/javastation/whitepapers

[vnc] The Olivetti & Oracle Research Lab: Virtual Network Computing
http://www.orl.co.uk/vnc/

OS

[apu] W. R. Stevens: Advanced Programming in the UNIX environment [Addison-Wesley]

[lkp] Michael Beck: Linux-Kernel-Programmierung [Addison Wesley]

[us] Helmut Herold: UNIX-Shells, 2. Auflage [Addison Wesley]

RFCs

[rfc640] J. Postel: Revised FTP Reply Codes

[rfc765] J. Postel: File Transfer Protocol

[rfc783] K. R. Sollins: The TFTP Protocol (Revision 2)

[rfc791] Internet Protocol

[rfc793] Transmission Control Protocol

[rfc821] J. B. Postel: Simple Mail Transfer Protocol

[rfc868] J. Postel: Time Protocol

[rfc913] Mark K. Lottor: Simple File Transfer Protocol

[rfc951] Bill Croft, John Gilmore: Bootstrap Protocol (BOOTP)

[rfc1013] Robert W. Scheifler: X Windows System Protocol, Version 11

[rfc1014] Sun Microsystems, Inc.: XDR: External Data Representation Standard

[rfc1034] P. Mockapetris: Domain Names - Concepts and Facilities

[rfc1094] Sun Microsystems: NFS: Network File System Protocol Specification

[rfc1332] G. McGregor: The PPP Internet Protocol Control Protocol (IPCP)

[rfc1334] B. Lloyd: PPP Authentication Protocols

[rfc1379] R. Braden: Extending TCP for Transactions - Concepts

[rfc1413] M. St. Johns: Identification Protocol

[rfc1497] J. Reynolds: BOOTP Vendor Information Extensions

[rfc1531] R. Droms: Dynamic Host Configuration Protocol

[rfc1631] K. Egevang, P. Francis: The IP Network Address Translator (NAT)

[rfc1644] R. Braden: T/TCP -- TCP Extensions for Transactions

[rfc1661] W. Simpson: The Point-to-Point Protocol (PPP)

[rfc1738] T. Berners Lee, L. Masinter, M. McCahill: Uniform Resource Locators (URL)

[rfc1794] T. Brisco: DNS Support for Load Balancing

[rfc1813] B. Callaghan: NFS Version 3 Protocol Specification

[rfc1866] T. Berners-Lee: Hypertext Mark-up Language - 2.0

[rfc1877] S. Cobb: PPP IPCP Extension for Name Server Addresses

[rfc1889] H. Schulzrinne: RTP: A Transport Protocol for Real-Time Applications

[rfc2002] C. Prekins: IP Mobility Support

[rfc2068] R. Fielding: Hypertext Transfer Protocol -- HTTP/1.1

[rfc2205] R. Braden: Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification

Security

[chr] Wietse Venema: chrootuid
ftp://ftp.win.tue.nl/pub/security/chrootuid/

[osr] Netscape: Object Signing Resources
http://developer.netscape.com/library/documentation/signedobj/

[pus] Simson Garfinkel: Practical UNIX & Internet Security [O'Reilly & Associates, Inc.]

[ssl] Netscape: Secure Socket Layer Protocol Version 3.0
http://www.netscape.com/newsref/std/SSL.html

[ssljt] Phaos Technology: SSL Java Toolkit
http://www.phaos.com/solutions.html

[tcpw] Wietse Venema: TCP Wrapper
ftp://ftp.win.tue.nl/pub/security/tcp_wrapper.txt.Z

[uisr] Cisco: Understanding Internet Security Risks
http://www.cisco.com/warp/public/751/centri/centr_wp.htm

Software Design

[Boo94] Grady Booch: Object-Oriented Analysis and Design with Applications [Benjamin/Cummings]

[dp] Erich Gamma & Co: Design Patterns [Addison-Wesley]

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorenson: Object-Oriented Modelling and Design [Prentice Hall]

[Rum94] James Rumbaugh: The life of an object model: How the object model changes during development [IEEE Transactions on Software Engineering, September 1984]