Network Working Group
INTERNET-DRAFT

S.E. Hardcastle-Kille
University College London
March 1992

The Simple OSI Stack

Status of this Memo

This document specifies a mapping of protocols defined by the Abstract Service Description Conventions onto Connection Oriented Transport Service, Connectionless Transport Service, TCP, and UDP [MHS88b]. This mapping will support all ROS (Remote Operation Service) based protocols, such as directory, both with and without use of RTS (Reliable Transfer Service), and all of the X.400 services [ISO88, CCI88, MHS88a].

Four example applications of the Simple OSI Stack (SOS) are given in appendices. These are all "real" examples, which will are quite likely to evolve into formal specifications. They are intended to show that specification of SOS applications is very straightforward and useful. These examples will be dropped from future versions of this document, and perhaps be replaced by artificial examples.

This draft document will be submitted to the RFC editor as a protocol standard. Distribution of this memo is unlimited. Please send comments to the author or to the discussion group <osi-ds@CS.UCL.AC.UK>.

Contents

1 Introduction **2**

2 Goals **2**

3 Model **4**

4 Deployment in Conjunction with Full Stack OSI **4**

5 Comparison with Previous Work **4**

6 PDUs **4**

6.1 Application Entity Titles 7

7 Operational Procedures **7**

7.1 Terminology 7

7.2 User Data 7

7.3 Binding 7

7.4 Unbinding 8

7.5 Operation 9

7.6 Reject 10

8 Mapping onto Used Services **10**

8.1 Connectionless Mappings 10

8.1.1 Connectionless Transport Service 10

8.1.2 UDP 10

8.2 Connnection Oriented Mappings 11

8.3 Connection Oriented Transport Service 11

8.3.1 TCP 11

9 Streaming **11**

10 Checkpointing and Resumption **12**

11 Application Level Relaying **13**

12 Encoding **14**

13 Addressing **14**

13.1 Application Context 14
13.2 COTS and CLTS Mappings 15
13.3 TCP and UDP Mappings 15

14 Security Considerations **15**

15 Author's Address **16**

A Nameserver Protocol 1 **16**
A.1 Options 17
A.2 Mapping 18
A.3 Encoding 18
A.4 Port Assignment 18

B Nameserver Protocol 2 **18**
B.1 Options 20
B.2 Mapping 20
B.3 Encoding 20
B.4 Port Assignment 20

C MTA Transfer **20**
C.1 Options 20
C.2 Mapping 20
C.3 Encoding 21
C.4 Port Assignment 21

D Directory Access Protocol **21**
D.1 Options 22
D.2 Mapping 22
D.3 Encoding 22
D.4 Port Assignment 23

1 Introduction

The majority of OSI Applications fall into two classes:

1. ROS based services without RTS.
2. ROS based services with RTS.

A third category is closely related: RTS based services which do not use ROS. There is only one well known protocol in this category (X.400 P1), but it is an important protocol.

This family of protocols uses the session and presentation layer service. Most of the services provided by these layer protocols are not used by this family of applications. These protocols use three Application Service Elements (ASEs): Association Control; Reliable Transfer; and Remote Operations. Much of the complexity of the protocols used to provide the ASE services upwards relate to the way in which the session and presentation services are used.

A much simpler mapping onto the transport service is possible for this family of protocols. This specification defines such a mapping.

There are a number of reasons why this work is needed:

- The potential for higher performance, and increasing the domain of applicability of ASDC based applications. In particular, it will make it much easier to achieve performance comparable to conventional RPC approaches.
- It reduces complexity, which is inherently desirable.
- It reduces the number of documents which an implementor needs to read.
- It enables a number of problems which have been exposed with ROS-based implementations to be fixed. These are discussed in the next section.

2 Goals

The following goals are defined for this specification. The first is the initial motivation, described above. The remaining goals are additional features, which are not available in the current mappings, but are seen to be desirable.

- To provide a mapping from the ASDC onto COTS, which is a strict replacement for the mapping onto ACSE, RTS, and ROS.

- To provide a mapping onto CLTS, which will be useful for some applications specified in ROS.
- To provide mappings onto TCP and UDP, the equivalent TCP/IP services.
- With ROS mapped onto RTS, there is a strict "layering". This makes the mapping awkward to implement in a manner such that transfer resumption can be used effectively, and also forces use of RTS for all ROS PDUs, some of which may be small. SOS allows checkpointing/resumption to be selected for some PDUs only, and does not require two way alternate.
- Rebinding may be done without breaking the transport connection. An example of the utility of this is for an application where an agent has bound with one authentication, and wishes to rebind with different authentication.
- A means for the responder of the association to request clean association release. An example of where this might be useful is when a DUA connects to a DSA, and holds the connection open. It is desirable that the DSA can request the DUA to cleanly close the association.
- A means for the responder of the association to reject a request to unbind is provided. This may be useful if the SOS-initiator requests the unbind at the same time as the SOS-responder invokes an operation.
- Provide an unambiguous means for an RTS user to permanently reject a user PDU (e.g., for protocol violation).
- Give support for "streaming". ROS-based specifications often define large PDUs (e.g., messages). If the protocol is being used interactively, it is desirable that the PDU is displayed as it is being transferred across the net. For example, when listing in the directory, it would be useful to request a very large number of entries, and then let them "flow" over the net. This would be a much cleaner way to provide the paged results mechanisms of X.500(92). This could in principle be implemented without change of protocol, but requires a very awkward interaction between the network, the application protocol, and the application. SOS defines an approach to allow the ROS PDUs to be broken on boundaries which have meaning to the application. This will make streaming straightforward to implement, whilst retaining the cleanliness of the existing ROS specification.
- Provide support for application level relaying. This is beneficial when direct connectivity is not possible at the network level either for physical or policy reasons. In the former case, this approach is much cleaner than transport bridging. In the latter, it provides a clean management model for control of external access.

3 Model

This specification essentially follows the OSI model. The services of the session and presentation layer required by these applications are conceptually provided by SOS. SOS follows the OSI approach to services, but performs layer compression to radically simplify the protocols.

4 Deployment in Conjunction with Full Stack OSI

For deployment to be effective, and not simply to add in another boundary it is important that server implementations support both SOS and full stack OSI, in order to provide for application relaying and overall uniform service.

5 Comparison with Previous Work

A similar piece of work was attempted earlier in the Lightweight Presentation Protocol (LPP) [Ros88]. Whilst this has been implemented and deployed, it has not achieved widespread success. Whilst there are some similarities to SOS, there are quite a few differences, some of which I believe to be critical:

- LPP cut at presentation level, which removed some potential for simplification within the application layer.
- LPP maps onto TCP only. SOS also maps onto COTS, and so can be used with OSI lower layers.
- LPP gives only a connection-oriented mapping. SOS also gives a connectionless mapping, which will be critical for some applications (e.g., Directory).
- LPP does not support ROS with RTS.
- LPP cannot support X.400. This is a critical problem.
- The LPP implementation in ISODE is a compile time choice, which prevents building relays to full-stack OSI. Whilst this might seem to be a detail, I believe that LPP would have definite local uses today if this had been available.

6 PDUs

This section defines the basic SOS PDUs. At the top level, SOS PDUs are broken into four types. The KernelPDU is described here, as this covers the basic approach of SOS. Other parts of SOS are defined later. A protocol

using SOS will often only need a small selection of the PDUs. This is defined implicitly by the ASDC description.

```
SOSPDU ::= CHOICE {
    KernelPDU,
    ResumptionPDU,
    StreamingPDU,
    RelayPDU }


KernelPDU ::= CHOICE {
    bind-argument      [0] BindArgument,
    bind-result        [1] BindArgument,
    bind-error         [2] OptionalUserData,
    unbind-argument    [3] UnbindArgument,
    unbind-result      [4] OptionalUserData,
    unbind-error       [5] OptionalUserData,
    operation-argument [6] Argument,
    operation-result   [7] Result,
    operation-error    [8] Error,
    reject             [9] Reject


UserData ::= ANY   -- defined by application --

OptionalUserData ::= SEQUENCE {UserData OPTIONAL}

BindArgument ::= SEQUENCE {
    version-number           [0] INTEGER,
    application-entity-title [1] DirectoryName OPTIONAL,
    application-context      [2] OBJECT IDENTIFIER OPTIONAL,
    association-identifier   [3] AssociationIdentifer OPTIONAL,
    resumption-information   [4] ResumptionInformation OPTIONAL,
    use-streaming            [5] BOOLEAN DEFAULT FALSE,
    user-data                [6] UserData OPTIONAL }


DirectoryName ::= CHOICE {
                        DistinguishedName,
                        IA5String  }

AssociationIdentifier ::= PrintableString

UnbindArgument ::= SEQUENCE {
    type ENUMERATED {
        normal(1),
        retain-transport-connection(2),
```

```
        responder-request(3) }
    UserData }


Argument ::= SEQUENCE {
        invoke-id InvokeID,
        linked-id [0] InvokeID,
        operation INTEGER,
        Userdata OPTIONAL }

InvokeID ::= INTEGER

Result ::= SEQUENCE {
        InvokeID,
        Userdata OPTIONAL }

Error ::= SEQUENCE {
        InvokeID,
        ErrorID,
        UserData }

ErrorID ::= INTEGER

Reject ::= SEQUENCE {
        reason ENUMERATED {
                temporary-provider-reject(0),
                temporary-user-reject(1),
                                -- permanent errors
                provider-protocol-error(2)
                user-protocol-error(3),
                user-permanent-reject(4),
                version-not-supported(5),
                duplicate-invokation(6),
                unrecognised-operation(7),
                unrecognised-linked-id(8),
                linked-response-unexpected(9),
                unrecognised-invokation(10),
                result-response-unexpected(11),
                unrecognised-invokation(12),
                error-response-unexpected(13),
                unrecognised-error(14),
                unexpected-error(15) }
        supplementary-information PrintableString OPTIONAL }
```

6.1 Application Entity Titles

An Application Entity Title (AET) is a directory distinguished name. SOS allows two different encodings of AET. An application using SOS will usually specify which form should be used. The forms are:

1. The encoding of Distinguished Name, as specified in the OSI Directory.
2. A string representation of Distinguished Name, as specified in [HK92a]. This is available, as the encoding of Distinguished Name is complex, and for many applications, it will be preferable to use this simple form.

7 Operational Procedures

7.1 Terminology

SOS-entity A SOS provider entity.

SOS-user The entity which is using the services provided by a SOS-entity.

SOS-use-definition The protocol specification which is mapping onto SOS.

SOS-initiator The SOS-entity which initiates the SOS Association.

SOS-reponder The SOS-entity which responds to the SOS association.

SOS-operation-invoker The SOS-entity which invokes an operation.

SOS-operation-responder The SOS-entity which responds to an invoked operation.

7.2 User Data

Many of the PDUs have an Element `UserData`. This will always refer to a type definition from the ASDC description. For example, in the `Operation` PDU, the type is defined in the OPERATION MACRO of ROS in the SOS-use-definition.

7.3 Binding

If a BIND definition is present in the SOS-use-definition ASDC specification, then the procedure for binding must be followed. If this definition is not present, the procedure may still be followed. The protocol mapping onto SOS should specify if this mandatory, optional, or absent.

When binding procedures are followed, a `bind-argument` is sent by the SOS-initiator, and either a `bind-result` or `bind-error` is returned by the SOS-responder. The parameters for `bind-argument` are as follows.

version This is set to 1.

application-entity-title The setting of this parameter, and the form used is defined by the SOS-use-definition.

association-identifier This is a unique identifier of the association between the pair of application entities. It will always be set if `resumption-information` is present, and may be set in other cases for logging purposes.

resumption-information This will be present if resumption is requested.

use-streaming This may be present if explicitly permitted by the SOS-use-definition.

user-data This is a value of the type defined in the BIND ARGUMENT.

If the SOS-responder rejects the bind, then a `bind-error` is sent, with the UserData omitted.

If the user of the SOS-responder rejects the bind, then a `bind-error` is sent, with the UserData type defined by the ERROR parameter.

If the user of the SOS-responder accepts the bind, then a `bind-result` is sent, with the following parameters:

version This is set to the highest version supported, which is less than or equal to that proposed. If a mutually convenient version cannot be reached, a reject is sent.

application-entity-title The setting of this parameter, and the form used is defined by the SOS-use-definition.

association-identifier This is omitted.

resumption-information This will be present if resumption is requested and has been accepted.

use-streaming This may be present if explicitly permitted by the SOS-use-definition, and this service was requested.

user-data This is a value of the type defined in the BIND RESULT.

7.4 Unbinding

Unbinding is a handshake initiated by the SOS-initiator, which will send an `unbind-argument`. The SOS-responder will send back either a `unbind-result` or `unbind-error`. This procedure is always followed if the SOS-use-definition ASDC contains an UNBIND.

When the SOS-initiator sends a `unbind-argument`, the UserData is defined by the UNBIND ARGUMENT. The `type` may have two values:

normal This is the usual case, and indicates that no more data will be given to the transport provider.

retain-transport-connection This indicates that a bind will follow the unbind.

If the user of the SOS-responder wishes to accept the unbind, it will send an `unbind-result`, with the optional UserData type defined by the UNBIND RESULT.

If the SOS-responder wishes to reject the unbind it will send an `unbind-error`, with no UserData.

If the user of the SOS-responder wishes to reject the unbind it will send an `unbind-error`, with the UserData type defined by the UNBIND ERROR.

If the SOS-responder wishes to end the association, it should send an `unbind-error`, with the verb—type— set to responder-request. The SOS-initiator shall then initiate an unbind in the usual way, unless it has good reason not to (e.g., it has just started another operation).

7.5 Operation

Whether the SOS-initiator, the SOS-responder, or both may invoke operations is specified in the SOS-use-definition.

When a SOS-user issues an invoke, an `operation-argument` PDU is sent. The operation from the SOS-use-definition is identified by the `operation` parameter. All SOS operations must be identified by integers, not object identifiers. This does not to lose generality, as the operations only need to be unique in the context of a given application context. The UserData defined by the OPERATION ARGUMENT of the operation in question. The InvokeID is an integer, which uniquely identifies the operation. The value of this starts at 1, and monotonically increases of the life of the association.

The `operation-argument` is received by the SOS-operation-responder, and is passed to the associated SOS-user. The SOS-use-definition indicates the procedure for the SOS-user to follow. This may result, in a result, an error, or no response.

If a result is sent, the SOS-operation-responder sends a `operation-result`. The InvokeID matches the invoke, and the UserData type is defined by the OPERATION RESULT of the operation in question.

If an error is sent, the SOS-operation-responder sends a `operation-error`. The InvokeID matches the invoke, and the `error-id` identifies the error from the SOS-use-definition ASDC. The UserData type is defined by the PARAMETER of the ERROR.

7.6 Reject

A general purpose reject is provided, which can deal with user and provider errors in operations, bind, and unbind.

8 Mapping onto Used Services

8.1 Connectionless Mappings

The SOS PDUs are self delimiting. A sequence of SOS PDUs is mapped onto a connectionless PDU by simple concatenation, and no further encapsulation or parameters.

The SOS-initiator sends a PDU containing at least one SOS PDU. This will be:

- A `bind-argument`. In addition to what has been described above, parameters are set as follows:

  association-identifier This is a unique handle to identify the request.

  resumption-information Always absent.

  use-streaming Always false.

- Zero or more `operation-arguments`.
- An optional `unbind-argument`.

In all cases, the SOS-responder will send back a response. This will be sent to the called address of the SOS-initiator.

- A response to the bind.
- Responses to each operation that needs a response.
- A response to the unbind, if present in the request.

If no response is received, the SOS-Initiator may retransmit the request.

8.1.1 Connectionless Transport Service

The PDU containing the SOS PDUs is mapped onto A-UNIT-DATA.

8.1.2 UDP

The PDU containing the SOS PDUs is mapped onto a UDP Datagram.

8.2 Connnection Oriented Mappings

A connection is first established. If the bind procedure used, this handshake should take place before the SOS-initiator invokes any operations. Then operations may be invoked in either direction.

Where operations may be invoked by the SOS-responder, the unbind procedure must always be used. This will allow the SOS-responder to reject the unbind where operations are outstanding.

If an unbind is not accepted, the association is considered to be open, and the SOS-initiator should retry the unbind.

8.3 Connection Oriented Transport Service

Each PDU is mapped onto T-DATA.

The unbind procedure must always be used. When the unbind ends the transport connection, the SOS-initiator shall close the transport connection when it receives the `unbind-result`. When the SOS-responder has sent the `unbind-result`, it should wait for 10 seconds before closing the transport connection. This timer is to prevent the `unbind-result` from being flushed as the connection is closed.

8.3.1 TCP

Each PDU, which is self delimiting, is sent onto the TCP byte stream.

The TCP connection should always be closed cleanly.

9 Streaming

*** repeat text from intro ***

Use of streaming is a facility which must be requested by both SOS-entities. An SOS-use-definition may make use of streaming mandatory, optional, or absent. The facility is negotiated at bind time.

A SOS-use-definition defines which arguments, results and errors that streaming can be applied to, and the list of permitted breakpoints. Streaming is selected by omitting the UserData to be streamed from the PDU which would have contained it. This data is then sent in a sequence of streamed PDUs. There shall be a maximum of one streamed PDU being transmitted in each direction at any one time. This does not prohibit other (non-streamed) PDUs from being transferred. In particular, an abandon may be sent, to terminate the operation in mid-flow. The StreamingPDU is defined as follows:

```
StreamingPDU ::= CHOICE {
```

```
    first-or-intermediate [10] UserData,
    last [11] UserData }
```

The data is sent as a sequence of StreamingPDUs, which can be reconstructed into the original PDU. The individual PDUs should break at natural points, and thus be straightforward for the application to handle. The application will also need to define the ordering rules for sending such PDUs. In many cases, the first PDU can be the existing PDU, and then subsequent PDUs transfer bulky or repetitive information. Usually these PDUs will be a small component of the existing PDU.

If the operation is abandoned (perhaps at the request of the invoker) before all of the data is transferred, a Reject PDU should be sent, with reason `temporary-user-reject`.

10 Checkpointing and Resumption

```
ResumptionInformation ::= SEQUENCE {
        transfer-direction ENUMERATED {
                monologue-push(1),
                monologue-pull(2),
                two-way-simultaneous(3) },
        window-size INTEGER,
        checkpoint-size INTEGER,
        association-identifier [1] AssociationID OPTIONAL,
        forward [2] ResumptionPoint OPTIONAL,
        reverse [3] ResumptionPoint OPTIONAL }

ResumptionPoint       ::= INTEGER

ResumptionPDU ::= CHOICE {
    block     [12] BlockNumber,
    block-ack [13] INTEGER }

Block ::= SEQUENCE {
    block-number BlockNumber,
    last-block   BOOLEAN,
    data         OCTET STRING }

BlockNumber ::= INTEGER
```

When resumption is going to be used, ResumptionInformation is exchanged at bind time. The initiator proposes parameter settings, and then the responder replies. The values specified by the responder defines the agreement.

The responder may refuse resumption by omitting the entire parameter. Consider first the case when no transfer is being resumed. The parameters are as follows:

transfer-direction Three choices are offered: monologue in either direction, and `two-way-simultaneous`. The latter provides for simultaneous transfer in both directions. In monologue cases, there may be only one resumable transfer. In two way simultaneous, there may be up to one resumable transfer in each direction. Two way alternate may be provided by rebinding the association with the opposite monologue direction selected. The responder must accept the proposed setting, with the exception that two way simultaneous may be negotiated down to monologue in either direction.

window-size This defines the number of blocks which may be outstanding without an acknowledgement received. The responder may accept or decrease this value.

checkpoint-size This defines the block size, in units of 1024 bytes. With the exception of the last block, blocks must be exactly this size. This size is the user data, prior to wrapping in the OCTET STRING encoding. The responder may accept or decrease this value.

When initiating a new association, the remaining parameters are absent. When resuming a transfer, the first parameters must be exactly as negotiated in the transfer being resumed (i.e., the SOS-initiator must propose these parameters, and the SOS-responder accept them unchanged. The SOS-initiator must be the original SOS-initiator. The association is identified by the `association-identifier`. A transfer may be resumed in either direction or both. If the resumption point is not present, the transfer should be discarded, and started from scratch. The SOS-initiator proposes resumption points for each transfer. The SOS-responder may reduce these to a lower value. If the SOS-initiator is unable to resume at this point, it should rebind, and discard the transfers.

Note: The combined use of streaming and resumption is not currently allowed. It may be desirable to do this. This would mean allowing blocking on boundaries used for streaming. This may be desirable for some applications anyway.

An alternate approach might be to provide resumption for primitive components transmitted within a PDU (e.g., an OCTET STRING carrying a document within a directory access).

11 Application Level Relaying

```
RelayPDU ::= [14] CHOICE {
```

```
        directory-name [1] DirectoryName,
        presentation-address [2] PresentationAddress
        ip-address [3] IPAddress }

IPAddress ::= PrintableString -- dot notation
```

It is often not possible for applications to connect directly. Reasons for this include:

- The application services are mapped onto lower layer stacks which do not interwork.
- There are organisational policy reasons, which wish to exert application level controls at an organisational boundary.

Transport level bridging does not provide a solution to the second problem, and has a number of practical difficulties with the first.

A mechanism to provide straightforward application level relaying is provided. One or more relay PDUs may be sent at the start of the connection. These define a source route. The first RelayPDU indicates the next hop. The point may be defined by either its Directory Name, or presentation address.

12 Encoding

No encoding rules are defined in this protocol. The encoding rules to be used should be defined by the protocol which uses SOS. A protocol using SOS is considered to have a single abstract syntax, which includes the abstract syntax for SOS. The application then defines the transfer syntax to be used. An application may support multiple transfer syntaxes by defining an application context for each transfer syntax. This is done, so that new transfer syntaxes (standard or non-standard) may be introduced cleanly.

13 Addressing

13.1 Application Context

An application entity is identified by its directory name. When this is looked up in the directory, it will identify a set of supported application contexts.

13.2 COTS and CLTS Mappings

The location of an OSI Application Entity is defined by its presentation address. For SOS mappings, presentation and session selectors are not used. For mappings onto COTS or CLTS, the application context must be present in the `bind-argument`, except where the application entity only supports a single possible application context.

13.3 TCP and UDP Mappings

The location of a SOS-entity is defined by a new attributes:

```
IPApplicationEntity OBJECT CLASS
        SUBCLASS OF ApplicationEntity
        MUST CONTAIN IPAddress
        ::= oc-ip-application-entity

IPAddress ATTRIBUTE
        WITH SYNTAX IPAddress
        ::= ac-ip-address
```

The application context is implied by the selector, and so the application context is always omitted from the `bind-argument` and `bind-result`.

14 Security Considerations

Security is intentionally omitted from this version of SOS, although it is likely to be added in later versions. Some key protocols have applications specific security, and many useful things can be done without these additions. It is seen as desirable to progress these extensions separately.

The following security issues should be considered:

- Peer authentication at bind time.
- Proxy authentication, so that an entity indirectly using the association can be authenticated (e.g., to provide a shared DUA).
- Per-operation authentication (e.g., as done by directory).
- Data confidentiality.

References

[CCI88] The Directory — overview of concepts, models and services, December 1988. CCITT X.500 Series Recommendations.

[HK92a] S.E. Hardcastle-Kille. A string representation of distinguished name. Request for Comments in preparation, Department of Computer Science, University College London, January 1992.

[HK92b] S.E. Hardcastle-Kille. Using the OSI directory to achieve user friendly naming. Request for Comments in preparation, Department of Computer Science, University College London, January 1992.

[ISO88] Remote operations: Model, notation and service definition, December 1988. ISO/CCITT.

[MHS88a] CCITT recommendations X.400 / ISO 10021, April 1988. CCITT SG 5/VII / ISO/IEC JTC1, Message Handling: System and Service Overview.

[MHS88b] CCITT recommendations X.407/ ISO 10021, April 1988. CCITT SG 5/VII / ISO/IEC JTC1, Message Handling: Abstract Service Definition Conventions.

[Ros88] M.T. Rose. ISO Presentation Services on top of TCP/IP-based internets. Request for Comments 1085, The Wollongong Group, December 1988.

15 Author's Address

Steve Hardcastle-Kille
Department of Computer Science
University College London
Gower Street
WC1E 6BT
England

Phone: +44-71-380-7294

EMail: S.Kille@CS.UCL.AC.UK

DN: CN=Steve Hardcastle-Kille, OU=Computer Science,
O=University College London, C=GB

UFN: S. Hardcastle-Kille, Computer Science
University College London, GB

A Nameserver Protocol 1

This appendix gives an example nameservice protocol, which is a strict subset of X.500 Directory Access Protocol mapped onto SOS.

```
DAPNS DEFINITIONS ::=
BEGIN

IMPORTS SimpleCredentials, Name, EntryInformationSelection,
   CommonArguments, EntryInformation, CommonResults
   FROM X.511

IMPORTS attributeError, nameError, serviceError, securityError ;
   FROM X.519

DAPNSBind ::= BIND
    ARGUMENT  DAPNSBindArgument
    RESULT DAPNSBindResult
    ERROR DAPNSBindError

DAPNSBindResult ::= DAPNSBindArgument

DAPNSBindArgument ::= SET {
    [0] Credentials OPTIONAL }

Credentials ::= SimpleCredentials

read OPERATION
    ARGUMENT DAPNSReadArgument
    RESULT DAPNSReadResult
    ERROR {attributeError, nameError,
           serviceError, securityError }
    ::= 1

DAPNSReadArgument ::= SET {
    [0] Name,
    [1] EntryInformationSelection DEFAULT {},
    COMPONENTS OF CommonArguments }

DAPNSResult ::= SET {
    [0] EntryInformation,
    COMPONENTS OF CommonResults }

END
```

A.1 Options

Application Entity Titles should be omitted from the bind-argument, as the same information is carried at the application level.

Resumption and streaming are not used.

A.2 Mapping

Connectionlness and connection oriented mappings are provided. The following application contexts are defined:

```
ac-dap-ns-cots ::=
ac-dap-ns-clts ::=
ac-dap-ns-tcp ::=
ac-dap-ns-udp ::=
```

A.3 Encoding

The protocol is encoded using ASN.1 Basic Encoding Rules (BER).

A.4 Port Assignment

TCP PORT nnn is assigned UDP PORT nnn is assigned

B Nameserver Protocol 2

This appendix gives an example nameservice protocol which uses the X.500 data model, but provides an entirely new access protocol.

```
SimpleNameServerProtocol DEFINITIONS ::=
BEGIN

NSRead OPERATION
       ARGUMENT NSReadArgument
        RESULT NSReadResult
        ERRORS {NameError, UFNError, ServiceError}
        ::= op-ns-read

NSReadArgument ::= SEQUENCE {
        name IA5String,
        ufn BOOLEAN,
        types SET OF AttributeType }

NSReadResult ::=  SEQUENCE {
        distinguished-name IA5String,
        aliases-dereferenced INTEGER,
        SET OF ATTRIBUTE }

Attribute ::= SEQUENCE {
        AttributeType,
        SET OF AttributeValue }
```

```
AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY -- defined by AttributeType

NameError ::= SEQUENCE {
        name-matched IA5String,
        components-matched INTEGER,
        aliases-dereferenced INTEGER
        ufn-choices SET OF IA5String }

ServiceError ::= ENUMERATED {
        unwilling-to-perform(0),
        security-error(1),
        busy(2),
        dsa-unavailable(3) }


END
```

This protocol is for DUA to DSA access. It is used to look up selected information associated with an object.

The argument to NSRead has the following options:

name The string form of the name being looked up.

ufn If this is false, the name is a string encoded distinguished name [HK92a]. If this is true, the name is a user Friendly Name (UFN) [HK92b], which should be resolved by the DSA.

types This is a list of attributes to be returned.

If the distinguished name is matched, or the UFN is resolved unambiguously, the attributes matched are returned. This may be zero or more attributes.

The operation may fail for a number of reasons. A service error gives some of those. If an entry is not located unambiguously, a name error is given. This has the following options:

name-matched This is the distinguished name of the longest portion matched.

components-matched This is a count of the number of components in the original name which led to this match.

aliases-dereferenced The number of aliases dereferenced to reach the point matched.

name-choices If UFN is selected, this may be used to give a set of ambiguous choices at the next level to be resolved.

B.1 Options

Bind should always be used, although no application level parameters are specified. Application Entity Title should always be present, primarily for logging purposes. The string form of encoding should be used.

Resumption and streaming are not used.

B.2 Mapping

Only a connectionlness mapping is provided. The following application contexts are defined:

```
ac-simple-ns-clts ::=
ac-simple-ns-udp ::=
```

B.3 Encoding

The protocol is encoded using ASN.1 Basic Encoding Rules (BER).

B.4 Port Assignment

UDP PORT nnn is assigned

C MTA Transfer

This appendix defines a mapping of X.400 MTA Transfer (P1) onto SOS.

A PDU of format `MTS-APDU`, as defined in X.419 should be carried as the argument of an invoke only. There are no results or errors. The BIND arguments are defined in X.411 (MTA Abstract Service). There is no unbind.

C.1 Options

Application Entity Title should always be present in the bind, and the ASN.1 form should be used.

Resumption may be used, and streaming is not used.

C.2 Mapping

Only a connection oriented mapping is provided. The following application contexts are defined:

```
ac-sos-mts-transfer-cots ::=
ac-sos-mts-transfer-tcp ::=
```

C.3 Encoding

The protocol is encoded using ASN.1 Basic Encoding Rules (BER).

C.4 Port Assignment

TCP PORT nnn is assigned

D Directory Access Protocol

This appendix defines a mapping of X.500 Directory Access Protocol (DAP) onto SOS, which illustrates how a mapping which utilises streaming should be defined.

The standard DAP is used, with streaming allowed for the following:

- Breaking up search and list results so that (groups of) entries may be returned separately.
- Breaking up read result, so that attributes may be returned separately. Small attributes might be returned first, with large attributes such as photos held back.
- Very large attribute values may be broken up, so the the protocol can be used to retrieve things such as documents.

Streaming may not be used in conjunction with per-operation authentication or with uncorrelated results. The three following PDUs are allowed. Each define the PDUs allowed in the streamed result.

```
ListResult SOSDAPListResult ::= CHOICE {
           wrapper [0] ListResult,
           subordinates [1] SET OF SEQUENCE {
               RelativeDistinguishedName,
               aliasEntry [0] BOOLEAN DEFAULT FALSE,
               fromEntry [1] BOOLEAN DEFAULT TRUE
           }
    }
```

The element `wrapper` is sent first, and this uses the standard DAP PDU to send common information. Subsequent PDUs are sent as `subordinates`, which give additional results from the list. The ASN.1 of this PDU is the same as that used within ListResult.

```
SearchResult SOSDAPSearchResult ::= CHOICE {
           wrapper [0] SearchResult,
```

```
        hits [1] SET OF EntryInformation,
        poq [2] PartialOutcomeQualifier
    }
```

The element `wrapper` is sent first, and this uses the standard DAP PDU to send common information. Subsequent PDUs are sent as `hits`, which give additional results from the search. The PDU `poq` is used to indicate areas of the DIT which have not been searched. This can be repeated, and mixed with `hits`.

```
ReadResult SOSDAPReadResult ::= CHOICE {
        wrapper [0] ReadResult,
        attributes [1] SET OF Attribute,
        attribute-fragment [2] Attribute
    }
```

The element `wrapper` is sent first, and this uses the standard DAP PDU to send common information. Subsequent PDUs are sent as `attributes`, which give additional attributes from the entry. A single attribute may be fragmented, and this must be the last attribute sent (multiple attributes could be supported by addition of another PDU). Only attributes with a primitive syntax can be fragmented. The resulting attribute values from the `attribute-fragment` are concatenated to make the single large value.

D.1 Options

Application Entity Titles should be omitted from the bind-argument, as the same information is carried at the application level.

Resumption is not used, and streaming may be used.

D.2 Mapping

Only a connection oriented mapping is provided. The following application contexts are defined:

```
ac-dap-streamed-cots ::=
ac-dap-streamed-tcp ::=
```

D.3 Encoding

The protocol is encoded using ASN.1 Packed Encoding Rules (PER).

D.4 Port Assignment

TCP PORT nnn is assigned