INTERNATIONAL STANDARD

ISO/IEC 9594-4

Third edition
1998-12-15

Information technology — Open Systems Interconnection — The Directory: Procedures for distributed operation

Technologies de l'information — Interconnexion de systèmes ouverts (OSI) — L'Annuaire: Procédures pour le fonctionnement réparti

Reference number
ISO/IEC 9594-4:1998(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.ch
Web www.iso.ch

Published by ISO in 2000

Printed in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 9594 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 9594-4 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 6, *Telecommunications and information exchange between systems*, in collaboration with ITU-T. The identical text is published as ITU-T Recommendation X.518.

This third edition cancels and replaces the second edition (ISO/IEC 9594-4:1995), of which it constitutes a minor revision.

ISO/IEC 9594 consists of the following parts, under the general title *Information technology — Open Systems Interconnection — The Directory*:

— *Part 1: Overview of concepts, models and services*

— *Part 2: Models*

— *Part 3: Abstract service definition*

— *Part 4: Procedures for distributed operation*

— *Part 5: Protocol specifications*

— *Part 6: Selected attribute types*

— *Part 7: Selected object classes*

— *Part 8: Authentication framework*

— *Part 9: Replication*

— *Part 10: Use of systems management for administration of the Directory*

Annexes A and D form a normative part of this part of ISO/IEC 9594. Annexes B, C, E and F are for information only.

Introduction

This Recommendation | International Standard part together with other Recommendations | International Standards, has been produced to facilitate the interconnection of information processing systems to provide directory services. A set of such systems, together with the directory information which they hold, can be viewed as an integrated whole, called the *Directory*. The information held by the Directory, collectively known as the Directory Information Base (DIB), is typically used to facilitate communication between, with or about objects such as application entities, people, terminals and distribution lists.

The Directory plays a significant role in Open Systems Interconnection, whose aim is to allow, with a minimum of technical agreement outside of the interconnection standards themselves, the interconnection of information processing systems:

- from different manufacturers;
- under different managements;
- of different levels of complexity; and
- of different ages.

This Recommendation | International Standard specifies the procedures by which the distributed components of the Directory interwork in order to provide a consistent service to its users.

This third edition technically revises and enhances, but does not replace, the second edition of this Recommendation | International Standard. Implementations may still claim conformance to the second edition. However, at some point, the second edition will not be supported (i.e. reported defects will no longer be resolved). It is recommended that implementations conform to this third edition as soon as possible.

This third edition specifies version 1 and version 2 of the Directory protocols.

The first and second editions also specified version 1. Most of the services and protocols specified in this edition are designed to function under version 1. When version 1 has been negotiated, differences between the services and between the protocols defined in the three editions are accommodated using the rules of extensibility defined in this edition of ITU-T Rec. X.519 | ISO/IEC 9594-5. However, some enhanced services and protocols, e.g. signed errors, will not function unless all Directory entities involved in the operation have negotiated version 2.

Implementors should note that a defect resolution process exists and that corrections may be applied to this part of this International Standard in the form of technical corrigenda. The identical corrections will be applied to this Recommendation in the form of Corrigenda and/or an Implementor's Guide. A list of approved technical corrigenda for this part of this International Standard can be obtained from the subcommittee secretariat. Published technical corrigenda are available from your national standards organization. The ITU-T Corrigenda and Implementor's Guides may be obtained from the ITU Web site.

Annex A, which is an integral part of this Recommendation | International Standard, provides the ASN.1 module for directory distributed operations.

Annex B, which is not an integral part of this Recommendation | International Standard, describes an example of distributed name resolution.

Annex C, which is not an integral part of this Recommendation | International Standard, describes authentication in the distributed operations environment.

Annex D, which is an integral part of this Recommendation | International Standard, provides the definitions of the ASN.1 information object classes introduced in this Directory Specification.

Annex E, which is not an integral part of this Recommendation | International Standard, illustrates knowledge maintenance.

Annex F, which is not an integral part of this Recommendation | International Standard, lists the amendments and defect reports that have been incorporated to form this edition of this Recommendation | International Standard.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

INFORMATION TECHNOLOGY – OPEN SYSTEMS INTERCONNECTION – THE DIRECTORY: PROCEDURES FOR DISTRIBUTED OPERATION

SECTION 1 – GENERAL

1 Scope

This Recommendation | International Standard specifies the behaviour of DSAs taking part in the distributed Directory application. The allowed behaviour has been designed so as to ensure a consistent service given a wide distribution of the DIB across many DSAs.

The Directory is not intended to be a general purpose database system, although it may be built on such systems. It is assumed that there is a considerably higher frequency of queries than of updates.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.200 (1994) | ISO/IEC 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model.*
- ITU-T Recommendation X.500 (1997) | ISO/IEC 9594-1:1998, *Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services.*
- ITU-T Recommendation X.501 (1997) | ISO/IEC 9594-2:1998, *Information technology – Open Systems Interconnection – The Directory: Models.*
- ITU-T Recommendation X.509 (1997) | ISO/IEC 9594-8:1998, *Information technology – Open Systems Interconnection – The Directory: Authentication framework.*
- ITU-T Recommendation X.511 (1997) | ISO/IEC 9594-3:1998, *Information technology – Open Systems Interconnection – The Directory: Abstract service definition.*
- ITU-T Recommendation X.519 (1997) | ISO/IEC 9594-5:1998, *Information technology – Open Systems Interconnection – The Directory: Protocol specifications.*
- ITU-T Recommendation X.520 (1997) | ISO/IEC 9594-6:1998, *Information technology – Open Systems Interconnection – The Directory: Selected attribute types.*
- ITU-T Recommendation X.521 (1997) | ISO/IEC 9594-7:1998, *Information technology – Open Systems Interconnection – The Directory: Selected object classes.*
- ITU-T Recommendation X.525 (1997) | ISO/IEC 9594-9:1998, *Information technology – Open Systems Interconnection – The Directory: Replication.*

– ITU-T Recommendation X.530 (1997) | ISO/IEC 9594-10:1998, *Information technology – Open Systems Interconnection – The Directory: Use of System management for Administration of the Directory.*

– ITU-T Recommendation X.680 (1997) | ISO/IEC 8824-1:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

– ITU-T Recommendation X.681 (1997) | ISO/IEC 8824-2:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification.*

– ITU-T Recommendation X.682 (1997) | ISO/IEC 8824-3:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*

– ITU-T Recommendation X.683 (1997) | ISO/IEC 8824-4:1998, *Information technology – Abstract Syntax Notation One (ASN.1): Parametrization of ASN.1 specifications.*

– ITU-T Recommendation X.880 (1994) | ISO/IEC 13712-1:1995, *Information technology – Remote Operations: Concepts, model and notation plus Technical Corrigendum 1 (1995).*

– ITU-T Recommendation X.880 (1994)/Amd. 1 (1995) | ISO/IEC 13712-1:1995/Amd. 1: 1996, *Information technology – Remote Operations: Concepts, model and notation – Amendment 1: Built-in operations.*

– ITU-T Recommendation X.881 (1994) | ISO/IEC 13712-2:1995, *Information technology – Remote Operations: OSI realizations – Remote Operations Service Element (ROSE) service definition.*

– ITU-T Recommendation X.881 (1994)/Amd. 1 (1995) | ISO/IEC 13712-2:1995/Amd. 1: 1996, *Information technology – Remote Operations: OSI realizations – Remote Operations Service Element (ROSE) service definition – Amendment 1: Mapping to A-UNIT-DATA and built-in operations.*

3 Definitions

For the purpose of this Recommendation | International Standard the following definitions apply:

3.1 OSI Reference Model Definitions

The following term is defined in ITU-T Rec. X.200 | ISO/IEC 7498-1:

– *application entity title.*

3.2 Basic Directory Definitions

The following terms are defined in ITU-T Rec. X.500 | ISO/IEC 9594-1:

a) *(the) Directory;*

b) *Directory Information Base.*

3.3 Directory Model Definitions

The following terms are defined in ITU-T Rec. X.501 | ISO/IEC 9594-2:

a) *access point;*

b) *alias;*

c) *distinguished name;*

d) *Directory Information Tree;*

e) *Directory System Agent (DSA);*

f) *Directory User Agent (DUA);*

g) *relative distinguished name.*

3.4 DSA Information Model definitions

The following terms are defined in ITU-T Rec. X.501 | ISO/IEC 9594-2:

a) *category;*

b) *commonly usable;*

c) *context prefix;*

d) *cross reference;*

e) *DIB fragment;*

f) *DSA information tree;*

g) *DSA Specific Entry (DSE);*

h) *DSE type;*

i) *immediate superior reference;*

j) *knowledge information;*

k) *knowledge reference category;*

l) *knowledge reference type;*

m) *naming context;*

n) *non-specific knowledge;*

o) *non-specific subordinate reference;*

p) *operational attribute;*

q) *reference path;*

r) *specific knowledge;*

s) *subordinate reference;*

t) *superior reference.*

3.5 Directory replication definitions

The following terms are defined in ITU-T Rec. X.525 | ISO/IEC 9594-9:

a) *attribute completeness;*

b) *shadowing operational binding;*

c) *subordinate completeness;*

d) *unit of replication.*

3.6 Distributed operation definitions

The following terms are defined in this Recommendation | International Standard:

3.6.1 base object: The object or alias entry that is the target for an operation as issued by the originator.

3.6.2 chaining: The generic term for uni-chaining or multi-chaining.

3.6.3 context prefix information: Operational and user information supplied by the superior DSA to the subordinate DSA in a RHOB regarding DIT vertices superior to the subordinate context prefix.

3.6.4 distributed name resolution: The process by which name resolution is performed in more than one DSA.

3.6.5 error: Information sent from the performer to the requester conveying a negative outcome of a previously received request.

3.6.6 hard error: A definite error which indicates that the operation cannot currently be performed without external intervention.

3.6.7 hierarchical operational binding (HOB): Relationship between two master DSAs holding naming contexts, one of which is immediately subordinate to the other, in which the superior DSA holds a subordinate reference to the subordinate DSA.

3.6.8 modification operations: These are the Directory Modify Operations, i.e. Modify Entry, Add Entry, Remove Entry and Modify DN.

3.6.9 multi-chaining: A mode of interaction in which a DSA processing a request itself sends multiple requests either in parallel or sequentially to a set of other DSAs.

3.6.10 multiple entry interrogation operations: These are the Directory Search Operations, i.e. List and Search.

3.6.11 name resolution: The process of locating an entry by sequentially matching each RDN in a purported name to a vertex of the DIT.

3.6.12 non-specific hierarchical operational binding (NHOB): Relationship between two master DSAs holding naming contexts, one of which is immediately subordinate to the other, in which the superior DSA holds a non-specific subordinate reference to the subordinate DSA.

3.6.13 NSSR decomposition: Decomposition of non-specific knowledge references into subrequests for other DSAs to pursue; these subrequests may be either chained to these DSAs by the DSA performing the decomposition, or a continuation reference identifying the DSAs may be returned to the requester for it to pursue, or the decomposing DSA may pursue some of the subrequests, leaving others unexplored for the requester to pursue.

3.6.14 operation progress: A set of values which denotes the extent to which name resolution has taken place.

3.6.15 originator: The DUA that has initiated a specific (distributed) operation.

3.6.16 performer: DSA receiving a request (i.e. to perform an operation).

3.6.17 procedure: An (informal) specification of how a DSA maps a given set of input arguments and its DSA information tree into a result.

NOTE – Input arguments and results may correspond to information received in a requested operation and information sent in a reply, or they may represent intermediate stages in the computation of a reply from a requested operation. In 14.2 the former variety of input arguments and results are termed external.

3.6.18 relevant hierarchical operational binding (RHOB): Either a HOB or a NHOB, depending on the context.

3.6.19 referral: An outcome which can be returned by a DSA which cannot perform an operation itself, and which identifies one or more other DSAs more able to perform the operation.

3.6.20 reply: A result or an error.

3.6.21 request: Information consisting of an operation code and associated arguments to convey a directory operation from a requester to a performer.

3.6.22 request decomposition: Decomposition of a request into subrequests for other DSAs to pursue; these subrequests may be either chained to these DSAs by the DSA performing the decomposition, or continuation references identifying the DSAs may be returned to the requester for it to pursue, or the decomposing DSA may pursue some of the subrequests, leaving others unexplored for the requester to pursue.

3.6.23 requester: A DUA or DSA sending a request to perform (i.e. invoke) an operation.

3.6.24 single entry interrogation operations: These are the Directory Read Operations, i.e. Read and Compare.

3.6.25 soft error: An error which may be transient, or which may indicate a localized problem, in which case the use of a different knowledge reference or access point may enable a result or hard error to be obtained.

3.6.26 subordinate DSA: Of the two DSAs sharing a HOB or a NHOB, the DSA holding the subordinate naming context.

3.6.27 subrequest: A request generated by request decomposition.

3.6.28 superior DSA: Of the two DSAs sharing a HOB or a NHOB, the DSA holding the superior naming context.

3.6.29 superior, subordinate DSA: Two master DSAs holding naming contexts, one of which is immediately subordinate to the other; the relationship between the two DSAs is managed explicitly via a HOB (or NHOB), or exists implicitly by virtue of the superior DSA holding a subordinate (or non-specific subordinate) reference to the subordinate DSA.

3.6.30 target object name: The name of an entry either to which the operation is to be directed at a particular stage of name resolution, or which is involved in the evaluation of the operation.

3.6.31 uni-chaining: A mode of interaction optionally used by a DSA which cannot perform an operation itself. The DSA *chains* by invoking an operation of another DSA and then relaying the outcome to the original requester.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply:

ASN.1	Abstract Syntax Notation One
DOP	Directory Operational Binding Management Protocol
DISP	Directory Information Shadowing Protocol
DMD	Directory Management Domain
DSE	DSA Specific Entry
HOB	Hierarchical Operational Binding
NHOB	Non-specific Hierarchical Operational Binding
NSSR	Non-specific Subordinate Reference
RHOB	Relevant Hierarchical Operational Binding

5 Conventions

With minor exceptions this Directory Specification has been prepared according to the "Presentation of ITU-T/ISO/IEC common text" guidelines in the Guide for ITU-T and ISO/IEC JTC 1 Cooperation.

The term "Directory Specification" (as in "this Directory Specification") shall be taken to mean ITU-T Rec. X.518 | ISO/IEC 9594-4. The term "Directory Specifications" shall be taken to mean the X.500-series Recommendations and all parts of ISO/IEC 9594.

This Directory Specification uses the term "1988 edition systems" to refer to systems conforming to the first (1988) edition of the Directory Specifications, i.e. the 1988 edition of the series of CCITT X.500 Recommendations and the ISO/IEC 9594:1990 edition. This Directory Specification uses the term "1993 edition systems" to refer to systems conforming to the second (1993) edition of the Directory Specifications, i.e. the 1993 edition of the series of ITU-T X.500 Recommendations and the ISO/IEC 9594:1995 edition. Systems conforming to this third edition of the Directory Specifications are referred to as "1997 edition systems".

This Directory Specification presents ASN.1 notation in the bold Helvetica typeface. When ASN.1 types and values are referenced in normal text, they are differentiated from normal text by presenting them in the bold Helvetica typeface. The names of procedures, typically referenced when specifying the semantics of processing, are differentiated from normal text by displaying them in bold Times. Access control permissions are presented in italicized Times.

If the items in a list are numbered (as opposed to using "–" or letters), then the items shall be considered steps in a procedure.

This Directory Specification defines directory operations using the Remote Operation notation defined in ITU-T Rec. X.880 | ISO/IEC 13712-1.

SECTION 2 – OVERVIEW

6 Overview

The Directory Abstract Service allows the interrogation, retrieval and modification of Directory information in the DIB. This service is described in terms of the abstract Directory object as specified in ITU-T Rec. X.511 | ISO/IEC 9594-3.

Necessarily, the specification of the abstract Directory object does not in any way address the physical realization of the Directory: in particular it does not address the specification of Directory System Agents (DSA) within which the DIB is stored and managed, and through which the service is provided. Furthermore, it does not consider whether the DIB is centralized, i.e. contained within a single DSA, or distributed over a number of DSAs. Consequently, the requirements for DSAs to have knowledge of, navigate to, and cooperate with other DSAs, in order to support the abstract service in a distributed environment is also not covered by the service description.

This Directory Specification specifies the refinement of the abstract Directory object, the refinement being expressed in terms of a set of one or more DSA objects which collectively constitute the distributed directory service.

In addition, this Directory Specification specifies the permissible ways in which the DIB may be distributed over one or more DSAs. For the limiting case where the DIB is contained within a single DSA, the Directory is in fact centralized; for the case where the DIB is distributed over two or more DSAs, knowledge and navigation mechanisms are specified which ensure that the whole of the DIB is potentially accessible from all DSAs that hold constituent entries.

Portions of the DIB may also be replicated in multiple DSAs. The protocols described in this Directory Specification allow the use of replicated information to improve the availability, performance and efficiency of the distributed directory service. The use of replicated information is, to some extent, under the user's control, through the use of service control options. The procedures described in this Directory Specification also indicate some of the opportunities for design optimizations when using the replicated information.

Additionally, request handling interactions are specified that enable particular operational characteristics of the Directory to be controlled by its users. In particular, the user has control over whether a DSA, responding to a directory inquiry pertaining to information held in other DSA(s), has the option of interrogating the other DSA(s) directly (chaining) or, whether it should respond with information about other DSA(s) which could further progress the inquiry (referral).

Generally, the decision by a DSA to chain or refer is determined by the service controls set by the user, and by the DSA's own administrative, operational or technical circumstances.

Recognizing that, in general, the Directory will be distributed, and that directory inquiries will be satisfied by an arbitrary number of cooperating DSAs which may arbitrarily chain or refer according to the above criteria, this Directory Specification specifies the appropriate procedures to be effected by DSAs in responding to distributed directory inquiries. These procedures will ensure that users of the distributed Directory service perceive it to be both user-friendly and consistent.

SECTION 3 – DISTRIBUTED DIRECTORY MODELS

7 Distributed Directory System Model

The Directory abstract service, as defined in ITU-T Rec. X.511 | ISO/IEC 9594-3, models the Directory as an object which provides a set of directory services to its users. Users of the Directory access its services through an access point. The Directory may have one or more access points and each access point is characterized by the services it provides and the mode of interaction used to provide these services.

Figure 1 illustrates the distributed directory model which will be used as the basis for specifying the distributed aspects of the directory. It illustrates the Directory as comprising a set of one or more DSAs.

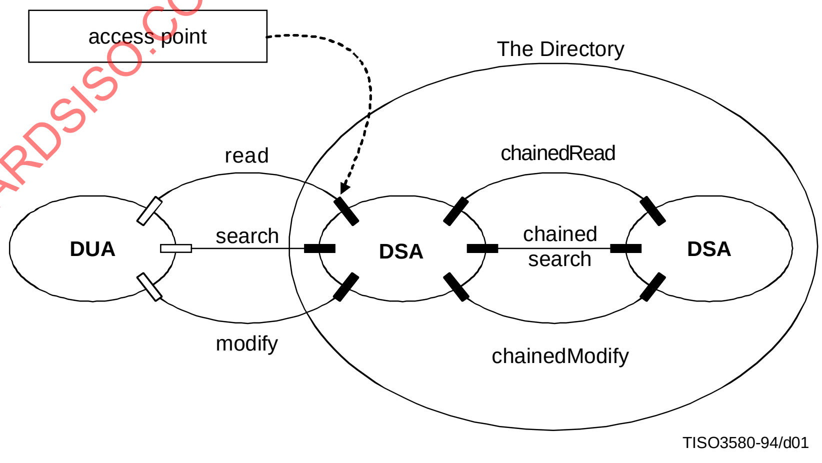


Figure 1 – Objects of the distributed Directory model

DSAs are specified in detail in the subsequent clauses of this Directory Specification. This clause merely states a number of their characteristics in order to serve as an introduction and to establish the relationship between this Directory Specification and the other Directory Specifications.

DSAs are defined in order that distribution of the DIB can be accommodated and that a number of physically distributed DSAs can interact in a prescribed, cooperative manner to provide directory services to the users of the directory (DUAs).

Figure 1 illustrates the relationship between the Directory abstract service and the DSA abstract service. The Directory abstract service defined in ITU-T Rec. X.511 | ISO/IEC 9594-3 is provided through a number of Directory operations. To realize this service, the DSAs that comprise the Directory interact with one another. The nature of this interaction is defined in terms of the service that one DSA may provide to another DSA, the DSA abstract service. The DSA abstract service is provided through a number of operations, termed chained operations, each having a counterpart in the Directory abstract service. Thus, a given operation in the Directory abstract service, e.g. Read, may require that the DSA providing the service interact with one or more other DSAs using chained operations, e.g. Chained Read.

8 DSA Interactions Model

A basic characteristic of the Directory is that, given a distributed DIB, a user should potentially be able to have any service request satisfied (subject to security, access control, and administrative policies) irrespective of the access point at which the request originates. In accommodating this requirement, it is necessary that any DSA involved in satisfying a particular service request have some knowledge (as specified in ITU-T Rec. X.501 | ISO/IEC 9594-2) of where the requested information is located and either return this knowledge to the requester or attempt to have the request satisfied on its behalf. (The requester may either be a DUA or another DSA: in the latter case both DSAs shall support the DSP.)

Three modes of DSA interaction are defined to meet these requirements, namely "uni-chaining", "multi-chaining", and "referral". Throughout the remainder of this Directory Specification, the generic term chaining is used to refer to uni-chaining and/or multi-chaining as appropriate to the context. "Chaining" refers to the attempt by a DSA to satisfy a request by sending one or more chained operations to other DSAs; "referral", to the return of knowledge information to the requester, which may then itself interact with the DSA(s) identified in the knowledge information.

Uni-chaining or a referral interaction may result from a single request. Alternatively, the request may be decomposed into several subrequests prior to the interaction. Multi-chaining or referral interactions, or a mixture of the two, may result from a decomposed request. Two types of decomposition are defined; NSSR decomposition and request decomposition.

8.1 Decomposition of a request

8.1.1 NSSR decomposition

NSSR decomposition is the process of preparing identical requests ready for transfer (either sequentially or in parallel) to several subordinate DSAs as a result of encountering an NSSR during name resolution. Non-specific subordinate references do not hold the RDNs of the referenced subordinate naming contexts, so the referencing DSA is unable to tell which subordinate DSA holds which subordinate naming context(s). During name resolution, a DSA encountering NSSRs shall send an identical request to each subordinate DSA (in the absence of shadowing). This may be done sequentially or in parallel. Typically, only one DSA will be able to continue with name resolution; the others will return the Service Error **unableToProceed**. In certain (rare) circumstances it is possible that more than one DSA will continue with name resolution, giving rise to duplicate results.

8.1.2 Request decomposition

Request decomposition, the other form of decomposing a request, is a process performed internally by a DSA prior to communication with one or more other DSAs. A request is decomposed into several, possibly different, subrequests such that each of the subrequests accomplishes a part of the original task. Request decomposition can be used only during operation evaluation of a List or Search. After request decomposition, each of the subrequests may then be chained to other DSAs to continue the task, or a partial result (an embedded referral) may be returned to the requester. An example of the same subrequest being generated to different DSAs is when an entry has subordinate references and/or NSSRs that together reference more than one DSA. An example of different subrequests being generated to the same or different DSAs is when two different entries are encountered during a Search (subtree), and each has a subordinate reference.

8.2 Uni-chaining

This mode of interaction (depicted in Figure 2) may be used by one DSA to pass on a request to another DSA when the former has knowledge about naming contexts held by the latter. Uni-chaining may be used to contact a single DSA pointed to in a cross reference, a subordinate reference, a superior reference, a supplier reference, or a master reference.

NOTE – In Figure 2, the order of interactions is defined by the numbers associated with the interaction lines.

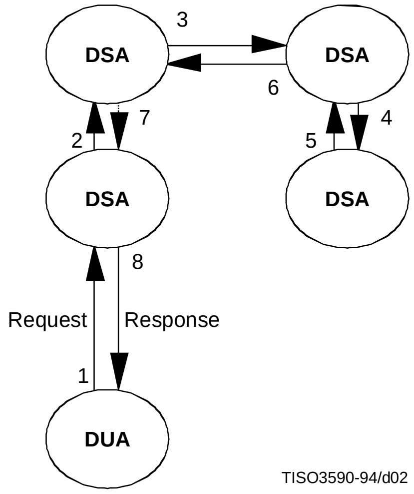


Figure 2 – Uni-chaining mode

8.3 Multi-chaining

This mode of interaction is used by a DSA for transferring several outgoing requests which have resulted from one incoming request, as a result of either request decomposition or NSSR decomposition.

8.3.1 Parallel multi-chaining

With parallel multi-chaining, the DSA transfers several outgoing requests simultaneously (see Figure 3a). Whilst parallel multi-chaining may give improved performance, it may under certain circumstances, e.g. in the presence of shadowing, cause duplicate results to be received.

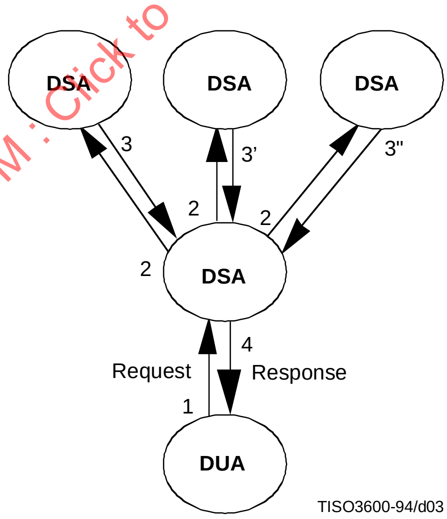
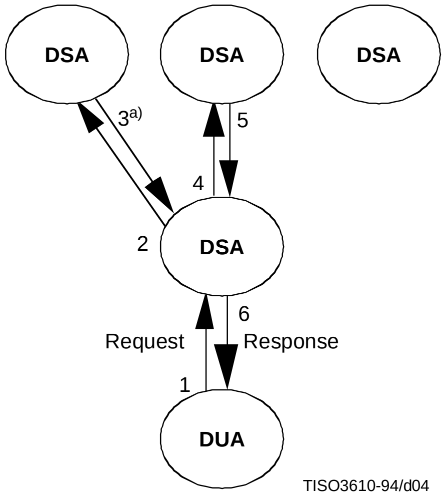


Figure 3a – Parallel Multi-chaining

8.3.2 Sequential multi-chaining

With sequential multi-chaining, the DSA transfers one outgoing request at a time and waits for the result or error of one request before sending the next (see Figure 3b). Whilst sequential multi-chaining may not be the quickest mode of interaction, it is unlikely that duplicate results will be received.

NOTE – A DSA may use a combination of parallel multi-chaining and sequential multi-chaining.



a) Unable to proceed.

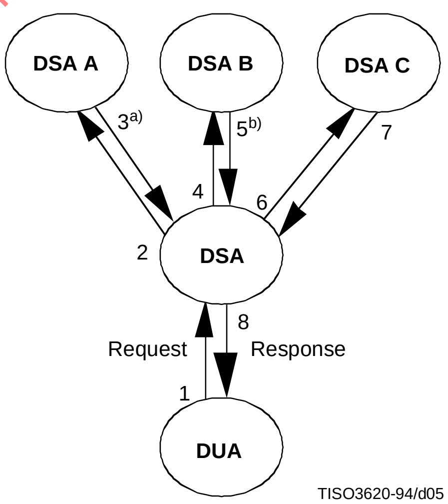
Figure 3b – Sequential Multi-chaining (as a result of NSSR decomposition)

8.4 Referral

A referral (depicted in Figures 4a and 4b) is returned by a DSA in response to a request from either a DUA or another DSA. The referral may constitute the whole response (in which case it is categorized as an error) or just part of the response. The referral contains a knowledge reference, which may be either a superior, subordinate, cross, non-specific subordinate, supplier, or master reference.

The DSA (Figure 4a) receiving the referral may use the knowledge reference contained therein, to subsequently chain or multi-cast (depending upon the type of reference) the original request to other DSAs. Alternatively, a DSA receiving a referral, may in turn pass the referral back in its response. A DUA (Figure 4b) receiving a referral may use it to contact one or more other DSAs to progress the request.

NOTE – In Figures 4a and 4b, the order of interactions is defined by the numbers associated with the interaction lines.



a) Referral to B
b) Referral to C

Figure 4a – Referral mode (DSA acts on referrals)

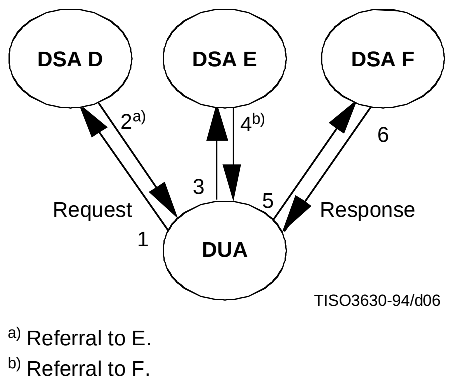


Figure 4b – Referral mode (DUA acts on referrals)

8.5 Mode Determination

If a DSA cannot itself fully resolve a request, it shall chain the request (or a request formed by decomposing the original one), to another DSA, unless:

a) chaining is prohibited by the user via the service controls, in which case the DSA shall return a referral or a **serviceError** with problem **chainingRequired**; or

b) the DSA has administrative, operational, or technical reasons for preferring not to chain, in which case the DSA shall return a referral.

NOTE 1 – A "technical reason" for not chaining is that the DSA identified in the knowledge reference does not support the DSP.

NOTE 2 – If the **localScope** service control is set, then the DSA (or DMD) shall either resolve the request or return an error.

NOTE 3 – If the user prefers referrals, the user should set **chainingProhibited**.

SECTION 4 – DSA ABSTRACT SERVICE

9 Overview of DSA Abstract Service

The service of the Directory is fully described in ITU-T Rec. X.511 | ISO/IEC 9594-3. When such a service is provided in a distributed environment, as modelled in clause 7, it can be regarded as being provided by means of a set of DSAs. This is illustrated in Figure 1.

For each operation defined in the Directory service, a corresponding "chained" operation is defined in the DSA abstract service for use between DSAs cooperating in the accomplishment of that Directory service operation. Thus, a DSA receiving a Read operation from a DUA might require the assistance of another DSA (e.g. a DSA holding the target entry or a copy of it) to satisfy it, and so send that DSA a Chained Read operation.

The information types exchanged in the DSA abstract service are defined in clause 10. The operations and errors of the DSA abstract service are defined in clauses 11 through 13.

10 Information types

10.1 Introduction

This clause identifies, and in some cases defines, a number of information types which are subsequently used in the definition of various of the operations of the DSA abstract service. The information types concerned are those which are common to more than one operation, are likely to be in the future, or which are sufficiently complex or self-contained to merit being defined separately from the operation which uses them.

Several of the information types used in the definition of the DSA abstract service are actually defined elsewhere. Subclause 10.2 identifies these types and indicates the source of their definition. Subclauses 10.3 through 10.9 identify and define an information type.

10.2 Information types defined elsewhere

The following information types are defined in ITU-T Rec. X.501 | ISO/IEC 9594-2:

- **aliasedEntryName**;
- **DistinguishedName**;
- **Name**;
- **RelativeDistinguishedName**.

The following information types are defined in ITU-T Rec. X.511 | ISO/IEC 9594-3:

(Bind)

- **DirectoryBind**

(Operations)

- **Abandon**

(Errors)

- **abandoned**;
- **attributeError**;
- **nameError**;
- **securityError**;
- **serviceError**;
- **updateError**.

(Information Object Class)

- **OPTIONALLY-PROTECTED**

(Data Type)

- **SecurityParameters**

The following information type is defined in ITU-T Rec. X.520 | ISO/IEC 9594-6:

- **PresentationAddress**.

10.3 Chaining Arguments

The **ChainingArguments** are present in each chained operation, to convey to a DSA the information needed to successfully perform its part of the overall task:

```
ChainingArguments ::= SET {
    originator              [0]   DistinguishedName OPTIONAL,
    targetObject            [1]   DistinguishedName OPTIONAL,
    operationProgress       [2]   OperationProgress
                                  DEFAULT { nameResolutionPhase notStarted },
    traceInformation        [3]   TraceInformation,
    aliasDereferenced       [4]   BOOLEAN DEFAULT FALSE,
    aliasedRDNs             [5]   INTEGER OPTIONAL,
                        -- only present in 1988 systems
    returnCrossRefs         [6]   BOOLEAN DEFAULT FALSE,
    referenceType           [7]   ReferenceType DEFAULT superior,
    info                    [8]   DomainInfo OPTIONAL,
    timeLimit               [9]   UTCTime OPTIONAL,
    securityParameters      [10]  SecurityParameters DEFAULT { },
    entryOnly               [11]  BOOLEAN DEFAULT FALSE,
    uniqueIdentifier        [12]  UniqueIdentifier OPTIONAL,
    authenticationLevel     [13]  AuthenticationLevel OPTIONAL,
    exclusions              [14]  Exclusions OPTIONAL,
    excludeShadows          [15]  BOOLEAN DEFAULT FALSE,
    nameResolveOnMaster     [16]  BOOLEAN DEFAULT FALSE,
    operationIdentifier     [17]  INTEGER OPTIONAL }
```

The various components have the meanings as defined below:

a) The **originator** component conveys the name of the (ultimate) originator of the request unless already specified in the security parameters. If **requester** is present in **CommonArguments**, this argument may be omitted.

NOTE 1 – Where the originator has alternative names differentiated by context, then the name used as the value of **originator** shall be the primary distinguished name, if known. Otherwise, authentication and access control based on the value of **originator** may not work as desired.

b) The **targetObject** component conveys the name of the object whose directory entry is being routed to. The role of this object depends on the particular operation concerned: it may be the object whose entry is to be operated on, or which is to be the base object for a request or subrequest involving multiple objects (e.g. **chainedList** or **chainedSearch**). This component can be omitted only if it has the same value as the object or base object parameter in the chained operation, in which case its implied value is that value.

Where the **targetObject** includes RDNs containing attribute type and value pairs for which there are multiple distinguished values differentiated by context, the RDNs that have been resolved shall be primary RDNs.

c) The **operationProgress** component is used to inform the DSA of the progress of the operation, and hence of the role which it is expected to play in its overall performance. The information conveyed in this component is specified in 10.5.

d) The **traceInformation** component is used to prevent looping among DSAs when chaining is in operation. A DSA adds a new element to trace information prior to chaining an operation to another DSA. On being requested to perform an operation, a DSA checks, by examination of the trace information, that the operation has not formed a loop. The information conveyed in this component is specified in 10.6.

e) The **aliasDereferenced** component is a **BOOLEAN** value which is used to indicate whether or not one or more alias entries have so far been encountered and dereferenced during the course of distributed name resolution. The default value of **FALSE** indicates that no alias entry has been dereferenced.

f) The **aliasedRDNs** component indicates how many of the RDNs in the **targetObject Name** have been generated from the **aliasedEntryName** attributes of one (or more) alias entries. The integer value is set whenever an alias entry is encountered and dereferenced. This component shall be present if and only if the **aliasDereferenced** component is **TRUE**.

NOTE 2 – This component is provided for compatibility with 1988 edition implementations of the Directory. DUAs (and DSAs) implemented according to later editions of the Directory Specifications shall always omit this parameter from the **CommonArguments** of a subsequent request. In this way, the Directory will not signal an error if aliases dereference to further aliases.

g) The **entryOnly** component is set to **TRUE** if the original operation was a search, with the subset argument set to **oneLevel** and an alias entry was encountered as an immediate subordinate of the **baseObject**. The DSA which successfully performs name resolution on the **targetObject** name, shall perform object evaluation on only the named entry.

h) The **returnCrossRefs** component is a Boolean value which indicates whether or not knowledge references, used during the course of performing a distributed operation, are requested to be passed back to the initial DSA as cross references, along with a result or referral. The default value of **FALSE** indicates that such knowledge references are not to be returned.

i) The **referenceType** component indicates, to the DSA being asked to perform the operation, what type of knowledge was used to route the request to it. The DSA may therefore be able to detect errors in the knowledge held by the invoker. If such an error is detected it shall be indicated by a **ServiceError** with the **invalidReference** problem. **ReferenceType** is described fully in 10.7.

NOTE 3 – If the **referenceType** is missing then the value **superior** shall be assumed.

j) The **info** component is used to convey DMD-specific information among DSAs which are involved in the processing of a common request. This component is of type **DomainInfo**, which is of unrestricted type:

```
DomainInfo ::=    ABSTRACT-SYNTAX.&Type
```

k) The **timeLimit** component, if present, indicates the time by which the operation is to be completed (see 16.1.4.1).

l) The **SecurityParameters** component is specified in ITU-T Rec. X.511 | ISO/IEC 9594-3. Its absence is deemed equivalent to there being an empty set of security parameters.

m) **authenticationLevel** component is optionally supplied when it is required to indicate the manner in which authentication has been carried out. The **AuthenticationLevel** data type is described in ITU-T Rec. X.501 | ISO/IEC 9594-2.

n) **uniqueIdentifier** component is optionally supplied when it is required to confirm the originator name. The **UniqueIdentifier** data type is described in ITU-T Rec. X.501 | ISO/IEC 9594-2.

o) The **entryOnly** component is set to **TRUE** if the original operation was a Search with the **subset** argument set to **oneLevel**, and an alias entry was encountered as an immediate subordinate of the **baseObject**. The DSA which successfully performs name resolution on the **targetObject** name shall perform object evaluation on only the named entry.

p) The **exclusions** component has significance only for Search operations; it indicates, if present, which subtrees of entries subordinate to the **targetObject** shall be excluded from the result of the Search operation (see 10.9).

q) The **excludeShadows** component has significance only for Search and List operations; it indicates that the search shall be applied to entries and not to entry copies. This optional component may be used by a DSA as one way to avoid the receipt of duplicate results (see 20.1).

r) The **nameResolveOnMaster** component only has significance during name resolution, and is only set if NSSRs have been encountered. If set to **TRUE**, it signals that subsequent name resolution, i.e. matching the remaining RDNs from **nextRDNToBeResolved**, shall not employ entry copy information; subsequent resolution of each remaining RDN shall be done in the master DSA for the entry identified by that RDN (see 20.1).

s) The **operationIdentifier** component facilitates the correlation of DAP operations with subsequent related DSP operations as well as with results. It is assigned by the DSA that first receives a DAP request or is copied from the chaining arguments of DSP requests that require further chaining. The DSA assigning the **operationIdentifier** shall not reuse the assigned integer for a sufficiently long time period. Correlation of related DAP and DSP requests and results is facilitated by a DSA logging, for each operation and result, the **operationIdentifier** together with the name of the DSA that assigned it (the first DSA in **traceInformation** on a chained request). Such correlation may be useful for the purposes of logging, auditing, charging and settlements, etc.

10.4 Chaining Results

The **ChainingResults** are present in the result of each operation and provide feedback to the DSA which invoked the operation.

```
ChainingResults ::= SET {
    info                 [0]   DomainInfo OPTIONAL,
    crossReferences      [1]   SEQUENCE OF CrossReference OPTIONAL,
    securityParameters   [2]   SecurityParameters DEFAULT { },
    alreadySearched      [3]   Exclusions OPTIONAL }
```

The various components have the meanings as defined below:

a) The **info** component is used to convey DMD-specific information among DSAs which are involved in the processing of a common request. This component is of type **DomainInfo**, which is of unrestricted type.

b) The **crossReferences** component is not present in the **ChainingResults** unless the **returnCrossRefs** component of the corresponding request had the value **TRUE**. This component consists of a sequence of **CrossReference** items, each of which contains a **contextPrefix** and an **accessPoint** descriptor (see 10.8).

```
CrossReference ::= SET {
    contextPrefix    [0]   DistinguishedName,
    accessPoint      [1]   AccessPointInformation }
```

A **CrossReference** may be added by a DSA when it matches part of the **targetObject** argument of an operation with one of its context prefixes. The administrative authority of a DSA may have a policy not to return such knowledge, and will, in this case, not add an item to the sequence.

c) The **SecurityParameters** data type is specified in ITU-T Rec. X.511 | ISO/IEC 9594-3. The absence of the **securityParameters** component is deemed equivalent to there being an empty set of security parameters.

d) The **alreadySearched** component, if present, indicates which subordinate RDNs subordinate to the **targetObject** have been processed as a part of a chained Search operation and therefore shall be excluded in a subsequent subrequest.

NOTE – Names in **contextPrefix** or **alreadySearched** shall be primary distinguished names and shall not contain alternative distinguished names.

10.5 Operation Progress

An **OperationProgress** value describes the state of progress in the performance of an operation which several DSAs shall participate in.

```
OperationProgress ::= SET {
        nameResolutionPhase          [0]    ENUMERATED {
                                                    notStarted          (1),
                                                    proceeding          (2),
                                                    completed           (3) },
        nextRDNToBeResolved          [1]    INTEGER OPTIONAL }
```

The various components have the meanings as defined below:

a) The **nameResolutionPhase** component indicates what phase has been reached in handling the **targetObject** name of an operation. Where this indicates that name resolution has **notStarted**, then a DSA has not hitherto been reached with a naming context containing the initial RDN(s) of the name. If name resolution is **proceeding**, then the initial part of the name has been recognized, although the DSA holding the target object has not yet been reached. The **nextRDNToBeResolved** indicates how much of the name has already been recognized [see 10.5 b)]. If name resolution is **completed**, then the DSA holding the target object has been reached, and performance of the operation proper is proceeding.

b) The **nextRDNToBeResolved** indicates to the DSA which of the RDNs in the **targetObject** name is the next to be resolved. It takes the form of an integer in the range one to the number of RDNs in the name. This component is only present if the **nameResolutionPhase** component has the value **proceeding**.

10.6 Trace Information

A **TraceInformation** value carries forward a record of the DSAs which have been involved in the performance of an operation. It is used to detect the existence of, or avoid, loops which might arise from inconsistent knowledge or from the presence of alias loops in the DIT.

```
TraceInformation ::= SEQUENCE OF TraceItem

TraceItem ::= SET {
        dsa                     [0]    Name,
        targetObject            [1]    Name  OPTIONAL,
        operationProgress       [2]    OperationProgress }
```

Each DSA which is propagating an operation to another adds a new item to the end of the sequence of **TraceItem**. Each such **TraceItem** contains:

a) the name of the DSA which is adding the item;

b) the **targetObject** name which the DSA adding the item received on the incoming request. This parameter is omitted if the request being chained came from a DUA (in which case its implied value is the **object** or **baseObject** in **XOperation**), or if its value is the same as the (actual or implied) **targetObject** in the **ChainingArgument** of the outgoing request;

c) the **operationProgress** which the DSA adding the item received on the incoming request.

dsa shall be the primary distinguished name and shall not contain alternative distinguished names. Each RDN in **targetObject** which has been processed shall be a primary RDN. Alternative distinguished values with contexts may be included within the **valuesWithContext** component of **AttributeTypeAndDistinguishedValue** in the RDN.

10.7 Reference Type

A **ReferenceType** value indicates one of the various kinds of reference defined in ITU-T Rec. X.501 | ISO/IEC 9594-2.

```
ReferenceType ::= ENUMERATED {
        superior                        (1),
        subordinate                     (2),
        cross                           (3),
        nonSpecificSubordinate          (4),
        supplier                        (5),
        master                          (6),
        immediateSuperior               (7),
        self                            (8) }
```

10.8 Access point information

There are three types of access points:

a) An **AccessPoint** value identifies a particular point at which access to the Directory, specifically to a DSA, can occur. The access point has a **Name**, that of the DSA concerned, and a **PresentationAddress**, to be used in OSI communications to that DSA.

```
AccessPoint ::= SET {
    ae-title                [0]   Name,
    address                 [1]   PresentationAddress,
    protocolInformation     [2]   SET OF ProtocolInformation OPTIONAL }
```

b) A **MasterOrShadowAccessPoint** value identifies an access point to the Directory. The **category**, either **master** or **shadow**, of the access point is dependent upon whether it points to a naming context or commonly usable replicated area.

```
MasterOrShadowAccessPoint ::= SET {
    COMPONENTS OF                 AccessPoint,
    category                [3]   ENUMERATED {
        master              (0),
        shadow              (1) } DEFAULT master } }
```

c) A **MasterAndShadowAccessPoints** value identifies a set of access points to the Directory, i.e. a set of related DSAs. These access points share the property that each refers to a DSA holding entry information from a common naming context (or a common set of naming contexts mastered in one DSA when the value is a value of the **nonSpecificKnowledge** attribute. A **MasterAndShadowAccessPoints** value indicates the **category** of each **AccessPoint** value it contains. The access point of the master DSA of the naming context need not be included in the set.

```
MasterAndShadowAccessPoints ::= SET OF MasterOrShadowAccessPoint
```

An **AccessPointInformation** value identifies one or more access points to the Directory.

```
AccessPointInformation ::= SET {
    COMPONENTS OF             MasterOrShadowAccessPoint,
    additionalPoints     [4]  SET OF MasterOrShadowAccessPoint OPTIONAL }
```

In the case of 1988 edition DSAs producing an **AccessPointInformation** value, the optional component of the set is absent. In the case of 1988 edition DSAs interpreting an **AccessPointInformation** value, any **MasterAndShadow-AccessPoints** values present are ignored.

In the case of post-1988 edition DSAs, the **MasterOrShadowAccessPoint** value component produced for an **AccessPointInformation** value may be of category master or shadow, as determined by the knowledge selection procedure of the DSA producing the value. It may be viewed as a suggested access point provided by the DSA generating the value to the DSA receiving it. A set of **MasterAndShadowAccessPoints** values may optionally also be produced for an **AccessPointInformation** value. This constitutes additional information which may be employed by the receiving DSA's knowledge selection procedure to determine an alternative access point.

10.9 Exclusions

As defined in 10.3, the **exclusions** component of **ChainingArguments** is used to limit the scope of a Search operation by identifying a number of entries subordinate to the target object which, together with all of their subordinates, shall not be included in the processing of a Search operation. The **exclusion** component is defined as a value of the ASN.1 type **Exclusions**.

```
Exclusions ::= SET OF RDNSequence
```

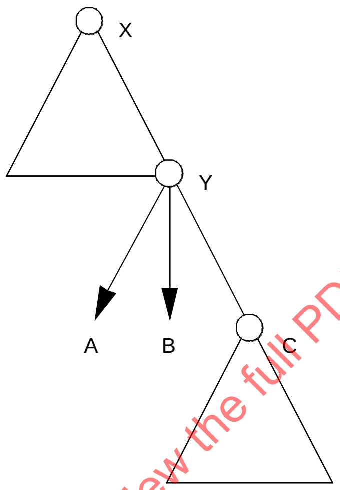
Each **RDNSequence** value in the **Exclusions** set should identify the context prefix of a naming context subordinate to the target object. If a DSA receives a **search** request with an **RDNSequence** value that does not conform to this constraint, the DSA may ignore that value. The **RDNSequence** is relative to the target object, and is not the distinguished name of the context prefix.

Exclusions shall be the primary distinguished names. Alternative distinguished names and context information may also be included.

Exclusions can, besides being part of a user request, be used by DSAs to minimize duplicate information returned from Search subrequests performed in the presence of shadowed information.

Figure 5 illustrates an example of the use of **Exclusions**. In this example, a DSA holds two replicated areas, one beneath the other. One starts with context prefix X, the other with context prefix C. An entry copy at Y has three subordinate references to naming contexts, A, B and C.

If, as an example, a subtree Search is performed in this DSA, starting with a base object within naming context X, the DSA can provide information from replicated areas X and C. The information from naming contexts A and B has to be provided via the subordinate references. When performing request decomposition, continuation references, to be used in either **partialResults** or chaining, will specify Y as the target object and C as a single element of an **Exclusions** set.



TISO3640-94/d07

Figure 5 – Exclusions

10.10 Continuation Reference

A **ContinuationReference** describes how the performance of all or part of an operation can be continued at a different DSA or DSAs. It is typically returned as a referral when the DSA involved is unable or unwilling to propagate the request itself.

```
ContinuationReference  ::=  SET {
        targetObject             [0]    Name,
        aliasedRDNs              [1]    INTEGER OPTIONAL, -- only present in 1988 systems
        operationProgress        [2]    OperationProgress,
        rdnsResolved             [3]    INTEGER OPTIONAL,
        referenceType            [4]    ReferenceType,
        accessPoints             [5]    SET OF AccessPointInformation,
        entryOnly                [6]    BOOLEAN DEFAULT FALSE,
        exclusions               [7]    Exclusions OPTIONAL,
        returnToDUA              [8]    BOOLEAN DEFAULT FALSE,
        nameResolveOnMaster      [9]    BOOLEAN DEFAULT FALSE }
```

The various components have the meanings as defined below:

a) The **targetObject** component indicates the name which is proposed to be used in continuing the operation. This might be different from the name received in **targetObject** of the incoming request if, for example, an alias has been dereferenced, or the base object in a search has been located.

RDNs in **targetObject** shall be primary RDNs (for the RDNs already processed). Alternative distinguished values with context may be included.

b) The **aliasedRDNs** component indicates how many (if any) of the RDNs in the target object name have been produced by dereferencing an alias. The argument is only present if an alias has been dereferenced.

NOTE – This component is provided for compatibility with 1988 edition implementations of the Directory. DUAs (and DSAs) implemented according to later editions of the Directory Specifications shall always omit this parameter from the **CommonArguments** of a subsequent request. In this way, the Directory will not signal an error if aliases dereference to further aliases.

c) The **operationProgress** indicates the amount of name resolution which has been achieved, and which will govern the further performance of the operation by the DSAs named, should the DSA or DUA receiving the **ContinuationReference** wish to follow it up.

d) The **rdnsResolved** component value (which need only be present if some of the RDNs in the name have not been the subject of full name resolution, but have been assumed to be correct from a cross reference) indicates how many RDNs have actually been resolved, using internal references only.

e) The **referenceType** component indicates what type of knowledge was used in generating this continuation.

f) The **accessPoints** component indicates the access points which are to be contacted to achieve this continuation. Only where non-specific subordinate references are involved can there be more than one **AccessPointInformation** item.

g) The **entryOnly** component is set to **TRUE** if the original operation was a search, with the **subset** argument set to **oneLevel**, and an alias entry was encountered as an immediate subordinate of the **baseObject**. The DSA which successfully performs name resolution on the **targetObject** name, shall perform object evaluation on only the named entry.

h) The **exclusions** component identifies a set of subordinate naming contexts that should not be explored by the receiving DSA.

i) The **returnToDUA** element is optionally supplied when the DSA creating the continuation reference wishes to indicate that it is unwilling to return information via an intermediate DSA (e.g. for security reasons), and wishes to indicate that information may be directly available via an operation over DAP between the originating DUA and the DSA. When **returnToDUA** is set to **TRUE**, **referenceType** may be set to **self**.

j) The **nameResolveOnMaster** element is optionally supplied when the DSA creating the continuation reference has encountered NSSRs. If set to **TRUE**, it signals that subsequent name resolution, i.e. matching the remaining RDNs from **nextRDNToBeResolved**, shall not employ entry copy information; subsequent resolution of each remaining RDN shall be done in the master DSA for the entry identified by that RDN (see 20.1).

11 Bind and Unbind

DSABind and **DSAUnbind**, respectively, are used by a DSA at the beginning and at the end of a period of accessing another DSA.

11.1 DSA Bind

A **DSABind** operation is used to begin a period of cooperation between two DSAs providing the Directory service.

```
DSABind ::= BIND
    ARGUMENT      DirectoryBindArgument
    RESULT        DirectoryBindResult
    BIND-ERROR    DirectoryBindError
```

The components of the **DSABind** are identical to their counterparts in the DirectoryBind (see ITU-T Rec. X.511 | ISO/IEC 9594-3) with the following differences:

– The **Credentials** of the **DirectoryBindArgument** allows information identifying the AE-Title of the initiating DSA to be sent to the responding DSA. The AE-Title shall be in the form of a Directory Distinguished Name.

– The **Credentials** of the **DirectoryBindResult** allows information identifying the AE-Title of the responding DSA to be sent to the initiating DSA. The AE-Title shall be in the form of a Distinguished Name.

– The DSA's name or AE-Title may use alternative distinguished names and may include context information.

NOTE 1 – Where names are used in either simple or strong credentials, it is possible to use alternative distinguished names, if they exist. However, authentication and access control based on the name may not work as desired if the primary distinguished name is not used. Following successful processing of an authenticated BIND operation, whatever the name used in the BIND argument, the bound entities shall thereafter know each other by their primary distinguished names, to facilitate operation of access controls while the BIND is in effect.

NOTE 2 – The credentials required for authentication may be carried by the Security Exchange Service Element (see ITU-T Rec. X.519 | ISO/IEC 9594-5) in which case they are not present in the bind arguments or results.

11.2 DSA Unbind

A **DSAUnbind** is used to end a period of cooperation between two DSAs providing the Directory service.

```
DSAUnbind ::= UNBIND
```

There are no arguments, results or errors.

12 Chained operations

For each of the operations used to access the Directory abstract service, there is an operation used between cooperating DSAs in a one-to-one correspondence. The names of the operations have been chosen to reflect that correspondence by prefixing the names of operations used between cooperating DSAs with the term "Chained".

The arguments, results, and errors of the chained operations are, with one exception, formed systematically from the arguments, results, and errors of the corresponding operations in the Directory abstract service (as described in 12.1). The one exception is the **ChainedAbandon** operation, which is syntactically equivalent to its Directory service counterpart (described in 12.2).

12.1 Chained operations

A DSA, having received an operation from a DUA, may elect to construct a chained form of that operation to propagate to another DSA. A DSA, having received a chained form of an operation, may also elect to chain it to another DSA. The DSA invoking a chained form of an operation may sign, encrypt, or sign and encrypt the argument of the operation; the DSA performing the operation, if so requested, may sign, encrypt, or sign and encrypt the result or error returned by the responder of the operation.

The chained form of an operation is specified using the parameterized type **chained { }**.

```
chained { OPERATION : operation } OPERATION ::= {
	ARGUMENT	OPTIONALLY-PROTECTED {
		SET {
			chainedArgument		ChainingArguments,
			argument		[0]	operation.&ArgumentType },
		DIRQOP.&dspChainedOp-QOP{@dirqop} }
	RESULT		OPTIONALLY-PROTECTED {
		SET {
			chainedResult		ChainingResults,
			result		[0]	operation.&ResultType },
		DIRQOP.&dspChainedOp-QOP{@dirqop} }
	ERRORS	{ operation.&Errors EXCEPT (referral | dsaReferral) }
	CODE		operation.&operationCode }
```

NOTE 1 – The operations of the Directory abstract service which may be used as the actual parameter of **chained { }** include the **abandoned** error. The presence of this error among the set of possible errors of a chained operation reflects the possibility discussed in 12.2, that a **ChainedAbandon** can be generated for a **ChainedModify** operation when a linked association fails.

NOTE 2 – The definitive specification of the DSA abstract service in Annex A applies this parameterized type to construct all the chained operations of the abstract service.

The argument of the derived operation has the components:

a) **chainedArgument** – This is a value of **ChainingArguments** which contains that information, over and above the original DUA-supplied argument, which is needed in order for the performing DSA to carry out the operation. This information type is defined in 10.3.

b) **argument** – This is a value **operation.&Argument** and consists of the original DUA-supplied argument, as specified in the appropriate clause of ITU-T Rec. X.511 | ISO/IEC 9594-3.

Should the request succeed, the result of the derived operation has the components:

a) **chainedResult** – This is a value of **ChainingResults** which contains that information, over and above that to be supplied to the originating DUA, which may be needed by previous DSAs in a chain. This information type is defined in 10.4.

b) **result** – This is a value **operation.&Result** and consists of the result which is being returned by the performer of this operation, and which is intended to be passed back in the result to the originating DUA. This information is as specified in the appropriate clause of ITU-T Rec. X.511 | ISO/IEC 9594-3.

Should the request fail, one of the errors of the set **operation.&Errors** will be returned, except that **dsaReferral** is returned instead of **referral**. The set of errors, which may be reported, is as described for the corresponding operation in ITU-T Rec. X.511 | ISO/IEC 9594-3. The error **dsaReferral** is described in 13.2.

12.2 Chained Abandon operation

A **chainedAbandon** operation is used by one DSA to indicate to another that it is no longer interested in having a previously invoked distributed operation performed. This may be for any of a number of reasons, of which the following are examples:

– the operation which led to the DSA originally chaining has itself been abandoned, or has implicitly been aborted by the breakdown of an association;

– the DSA has obtained the necessary information in another way, e.g. from a faster responding DSA involved in the parallel multi-chaining.

A DSA is never obliged to issue a **chainedAbandon**, or indeed to actually abandon an operation if requested to do so.

If **chainedAbandon** actually succeeds in stopping the performance of an operation, then a result will be returned, and the subject operation will return an **abandoned** error. If the **chainedAbandon** does not succeed in stopping the operation, then it itself will return an **abandonFailed** error.

12.3 Chained operations and protocol version

Operations which require a protocol version greater than v1 (such as the **modifyEntry** operation with certain arguments) or which return different results when used with a protocol version greater than v1 (such as **modifyEntry** with a signed argument) shall only be chained on associations with the same or a greater version number than that used to convey the request.

13 Chained errors

13.1 Introduction

For the most part, the same errors can be returned in the DSA abstract service which can be returned in the Directory abstract service. The exceptions are that the **dsaReferral** "error" is returned (see 13.2), instead of **Referral**, and the following service problems have the same abstract syntax but different semantics:

a) **invalidReference** – The DSA returning this error detected an error in the calling DSA's knowledge as specified in the **referenceType** chaining argument.

b) **loopDetected** – The DSA returning this error detected a loop in the knowledge information in the Directory.

The precedence of the errors which may occur is as for their precedence in the Directory abstract service, as specified in ITU-T Rec. X.511 | ISO/IEC 9594-3.

If an error occurs during a chained operation, the responding DSA may sign, encrypt, or sign and encrypt the error returned.

13.2 DSA Referral

The **dsaReferral** error is generated by a DSA when, for whatever reason, it doesn't wish to continue performing an operation by chaining the operation to one or more other DSAs. The circumstances where it may return a referral are described in 8.3. If the parameters of the operation were signed by the requestor, then the Directory applies the protection specified in the DIRQOP.

```
dsaReferral ERROR  ::=  {
        PARAMETER       OPTIONALLY-PROTECTED {
                SET {
                        reference           [0]     ContinuationReference,
                        contextPrefix       [1]     DistinguishedName OPTIONAL,
                        COMPONENTS OF CommonResults },
                DIRQOP.&dsaReferral-QOP{@dirqop} }
        CODE                    id-errcode-dsaReferral }
```

The various parameters have the meanings as described below:

a) The **ContinuationReference** contains the information needed by the invoker to propagate an appropriate further request, perhaps to another DSA. This information type is specified in 10.10.

b) If the **returnCrossRefs** component of the **ChainingArguments** for this operation had the value **TRUE**, and the referral is being based upon a subordinate or cross-reference, then the **contextPrefix** parameter may optionally be included. The administrative authority of any DSA will decide which knowledge references, if any, can be returned in this manner (the others, for example, may be confidential to that DSA).

A **contextPrefix** or a Continuation Reference shall be the primary distinguished name. Alternative distinguished values with context may be included within the **valuesWithContext** component of an **AttributeTypeAndDistinguishedValue** of any RDN.

SECTION 5 – DISTRIBUTED PROCEDURES

14 Introduction

14.1 Scope and Limits

This clause specifies the procedures for distributed operation of the Directory which are performed by DSAs. Each DSA individually performs the procedures described below; the collective action of all DSAs produces the full set of services provided to users by the Directory.

14.2 Conformance

The description of DSA procedures in this section is based on the models in clauses 8 and 9 of ITU-T Rec. X.501 | ISO/IEC 9594-2 and clauses 7 and 8. The flow charts and their corresponding textual descriptions are one means of mapping a given set of external (DAP and/or DSP) inputs to a DSA into one or more external outputs (i.e. a result, error, referral, or chained requests) produced by that DSA, depending on the particular DSA information tree held by that DSA.

It is probable that the Directory will be distributed across DSAs implemented according to different editions of the Directory Specifications, e.g. 1988, 1993 and 1997 editions. The DUA initiating the request will be unaware as to which edition the DSA or DSAs satisfying the DUA's request will have been implemented. Therefore to allow operation in such a heterogeneous environment, a DSA shall be implemented according to the rules of extensibility defined in clause 7 of ITU-T Rec. 519 | ISO/IEC 9594-5.

A DSA implementation shall be functionally equivalent to the external behaviour specified by these procedures described here. The algorithms used by a particular DSA implementation to derive the correct output(s) from the given inputs and DSA information tree held are not standardized.

NOTE – The flowcharts which accompany the procedures are intended to be used as aids towards understanding the procedures. They are not to be considered as being a precise alternative to the textual descriptions. Where there is a disparity between the textual description and the flowchart for a particular procedure, it is intended that the textual description take precedence.

14.2.1 Interaction between 1988 edition and 1988/1997 edition DSAs

If the modify operations evaluate across DSA boundaries (i.e. **addEntry** with **TargetSystem**, Remove or Rename a context prefix), then this Directory Specification only specifies how two 1997 edition DSAs or how a 1993 edition DSA and a 1997 edition DSA shall behave. The interaction between two 1988 edition DSAs, or between a 1988 edition DSA and a 1993/1997 edition DSA, is outside the scope of the Directory Specifications. When mixed edition DSAs have a hierarchical operational binding, knowledge of each other's edition may allow a consistent error to be given to the user.

14.3 Conceptual model

The complexity of the Directory's distributed operation gives rise to a need for conceptual modelling using both narrative and pictorial descriptive techniques. However, neither the narrative nor graphic diagrams should be construed as a formal description of distributed Directory operation.

14.4 Individual and cooperative operation of DSAs

The model views DSA operation from two separate perspectives, which, taken together, provide a complete, operational picture of the Directory.

a) **DSA-centered perspective** – In this perspective the set of procedures that support the directory is described from the viewpoint of a single DSA. This makes it possible to provide a definitive specification of each procedure and to fully account for their interrelationships and overall control structure. Clauses 16 through 22 describe the DSA procedures from a DSA-centered perspective.

b) **operation-centered perspective** – The DSA-centered view provides complete detail but makes it difficult to understand the structure of individual operations, which may undergo processing by multiple DSAs. Consequently clause 15 adopts a primarily operation-centered view to introduce the processing phases applicable to each.

To support the distributed operation of the Directory, each DSA shall perform actions needed to realize the intent of each operation and additional actions needed to distribute that realization across multiple DSAs. Clause 15 explores the distinction between these two kinds of actions. In clauses 16 through 22, both kinds of actions are specified in detail.

14.5 Cooperative agreements between DSAs

All DSAs which are in a subordinate/superior relationship due to the naming contexts that they hold, have hierarchical and/or non-specific hierarchical operational bindings between them, depending upon the types of knowledge reference held by the subordinate DSA.

Hierarchical and non-specific hierarchical operational bindings between DSAs may be administered using the procedures of clauses 24 and 25 of this Recommendation | International Standard, or by other means (e.g. telephone).

A DSA holding entries which are within the administrative area of its superior DSA shall administer the subschema and shall control access to the entries as required by the administrative authority. The regulation of entries within an administrative area may be performed as defined in ITU-T Rec. X.501 | ISO/IEC 9594-2 or may be by local mechanisms.

15 Distributed Directory behaviour

15.1 Cooperative fulfilment of operations

Each DSA is equipped with procedures capable of completely fulfilling all Directory operations. In the case that a DSA contains the entire DIB, all operations are, in fact, completely carried out within that DSA. In the case that the DIB is distributed across multiple DSAs, the completion of a typical operation is fragmented, with just a portion of that operation carried out in each of potentially many cooperating DSAs.

In the distributed environment, the typical DSA sees each operation as a transitory event: the operation is invoked by a DUA or some other DSA; the DSA carries out processing on the object and then directs it toward another DSA for further processing.

An alternative view considers the total processing experienced by an operation during its fulfilment by multiple, cooperating DSAs. This perspective reveals the common processing phases that apply to all operations.

15.2 Phases of operation processing

Every Directory operation may be thought of as comprising three distinct phases:

a) the *Name Resolution* phase in which the name of the object on whose entry a particular operation is to be performed is used to locate the DSA which holds the entry;

b) the *Evaluation phase* in which the operation specified by a particular directory request (e.g. a Read operation) is actually performed;

c) the *Results Merging phase* in which the results of a specified operation are returned to the requesting DUA. If a chaining mode of interaction was chosen, the Results Merging phase may involve several DSAs, each of which chained the original request or subrequest (as defined in 15.3.1 Request Decomposition) to another DSA during either or both of the preceding phases.

In the case of the operations Read, Compare, List, Search, Modify Entry, Modify DN and Remove Entry, name resolution takes place on the object name provided in the argument of the operation. In the case of Add Entry, name resolution's target entry is the immediately superior entry of that provided in the argument of the operation – it can be easily derived by removing the final RDN from the name provided in the operation argument. (This is done via local argument **m** in the **FindDSE** procedure of 18.3.1.)

An operation on a particular entry may initially be directed at any DSA in the Directory. That DSA uses its knowledge, possibly in conjunction with other DSAs, to process the operation through the three phases.

15.2.1 Name Resolution phase

Name Resolution is the process of sequentially matching each RDN in a purported Name to an arc (or vertex) of the DIT, beginning logically at the Root and progressing downwards in the DIT. However, because the DIT is distributed between arbitrarily many DSAs, each DSA may only be able to perform a fraction of the name resolution process. A given DSA performs its part of the Name Resolution process by traversing its local DSA information tree. This process is described in clause 18 and the accompanying diagrams (see Figures 9 through 12). Based on its local DSA information tree, and the knowledge information contained therein, a DSA is able to infer whether the resolution can be continued by one or more other DSAs, or whether the name is erroneous.

The Name Resolution phase is constrained to work within a DSA Information Tree if the **manageDSAIT** service control option is set.

15.2.2 Evaluation phase

When the Name Resolution phase has completed, the actual operation required (e.g. Read or Search) is performed.

Operations that involve a single entry interrogation – Read and Compare – may be carried out entirely within the DSA in which the entry is located.

Operations that involve multiple entries interrogation – List and Search – need to locate subordinates of the target, which may or may not reside in the same DSA. If they do not all reside in the same DSA, operations need to be directed to the DSAs specified in the subordinate, non-specific subordinate, supplier, or master references (as appropriate) to complete the evaluation process.

The Evaluation phase is constrained to work within a DSA Information Tree if the **manageDSAIT** service control option is set.

15.2.3 Results merging phase

The Results Merging phase is entered once some of the results of the Evaluation phase are available.

In those cases where the operation affected only a single entry, the result of the operation can simply be returned to the requesting DUA. In those cases where the operation has affected multiple entries on multiple DSAs, results can be combined. If protection is performed on the results, the results shall not be combined. The results should be returned to the DUA without performing merging.

The permissible responses returned to a requester after results merging include:

a) a complete result of the operation;

b) a result which is not complete because some parts of the DIT remain unexplored (applies to List and Search only). Such a *partial result* may include continuation references for those parts of the DIT not explored;

c) an error (a referral being a special case); and

d) if the requester was a DSA, a **ChainingResult**.

15.3 Managing Distributed Operations

Information is included in the argument of each operation which a DSA may be asked to perform indicating the progress of each operation as it traverses various of the DSAs of the Directory. This makes it possible for each DSA to perform the appropriate aspect of the processing required, and to record the completion of that aspect before directing the operation outward toward further DSAs.

Additional procedures are included in the DSA to physically distribute the operations and support other needs arising from their distribution.

15.3.1 Request Decomposition

Request decomposition is a process performed internally by a DSA prior to communication with one or more other DSAs. A request is decomposed into several subrequests such that each of the latter accomplishes a part of the original task. Request decomposition can be used, for example, in the search operation, after the base object has been found. After decomposition, each of the subrequests may then be uni-chained or multi-chained to other DSAs, to continue the task. If protection is performed on the arguments, request decomposition shall not be used.

15.3.2 DSA as Request Responder

A DSA that receives a request can check the progress of that request using the **operationProgress** parameter. This will determine whether the operation is still in the Name Resolution phase or has reached the evaluation phase, and what portion of the operation the DSA should attempt to satisfy. If the DSA cannot fully satisfy the request, it shall either pass (by uni-chaining or multi-chaining) the operation on to one or more DSAs which can help to fulfill the request, or return a referral to another DSA, or terminate the request with an error.

15.3.3 Completion of Operations

Each DSA that has initiated an operation or propagated an operation to one or more other DSAs shall keep track of that operation's existence until each of the other DSAs has returned a result or error, or the operation's maximum time limit has expired. This requirement applies to all operations, propagation modes and processing phases. It ensures the orderly closing down of distributed operations that have propagated out into the Directory.

15.4 Loop handling

The DIT may be in a state that can cause looping. As an example, looping can occur during name resolution where dereferencing one or more aliases brings the resolution back to the same branch of the DIT. Another potential cause of looping is through misconfigured knowledge references.

Within the context of a particular directory operation, a loop occurs if at any time the operation returns to a previous *state*, where state is defined by the following components:

– the name of the DSA currently processing the operation;

– the name of the **targetObject** as contained within the argument of the operation;

– the **operationProgress** as contained within the argument of the operation and as defined in 10.5.

This does not mean that an operation cannot be processed multiple times by a particular DSA. However, it does mean that the DSA will not process the same operation in the same state multiple times.

Looping is controlled using the **traceInformation** argument as defined in 10.6, which records the sequence of states a particular operation has gone through. Two strategies are defined to determine whether looping has occurred, or is about to occur. These are loop detection and loop avoidance, and they are described in 15.4.1 and 15.4.2 respectively.

Loop detection is mandatory and loop avoidance is optional.

15.4.1 Loop detection

On receipt of a directory operation a DSA shall initially validate the operation to ensure that it can be progressed. An important task of validation is to check for loops, by determining whether the current state of the operation appears in the sequence of previous states recorded in the **traceInformation** argument for that operation. This step of loop checking is loop detection.

15.4.2 Loop avoidance

Loop avoidance requires that a DSA, immediately prior to forwarding an operation to another DSA as part of a chaining procedure, determines whether the consequential state of the operation (which is the **traceItem** that the receiving DSA will add to **traceInformation** when it receives it) appears on the sequence of previous states recorded in the **traceInformation** argument for the original incoming operation.

In the case where referrals are received or acted upon, loop avoidance and loop detection cannot be achieved purely by examining **traceInformation**. In this case, each time a DSA acts on a referral, it needs to store the consequential state of the operation (i.e. the **traceItem** that the receiving DSA is going to add when it receives the request) along with a record of the incoming request. Before acting on or returning a referral, a DSA needs to check through this list, in order to check that an identical request has not been previously sent whilst trying to service the incoming operation.

15.5 Other considerations for distributed operation

15.5.1 Service controls

Some service controls need special consideration in the distributed environment in order that the operation is processed the way that was requested.

a) **chainingProhibited** – A DSA consults this service control when determining the mode of propagation of an operation. If it is set, then the DSA always uses referral mode. If, however, it is not set, the DSA can choose whether to use chaining or referral depending on its capabilities.

b) **timeLimit** – A DSA needs to take account of this service control to ensure that the time limit is not exceeded in that DSA. A DSA requested to perform an operation by a DUA, initially heeds the **timeLimit** expressed by the DUA as the available elapsed time in seconds for completion of the operation. If chaining is required, the **timeLimit** is included in the chaining argument to be passed to the next DSA(s). In this case, the same value of the limit is used for each chained request, and is the (UTC) time by which the operation shall complete to meet the originally specified constraint. On receiving **ChainingArguments** with a **timeLimit** specified, the receiving DSA respects this limit.

c) **sizeLimit** – A DSA needs to take account of this service control to ensure that the list of results does not exceed the size specified. The limit, as included in the common argument of the original request, is conveyed unchanged as the request is chained. If request decomposition is required, the same value is included in the argument to be passed to the next DSA, the full limit is used for each subrequest. When the results are returned, the requester DSA resolves the multiple results and applies the limit to the total to ensure that only the requested number are returned. If the limit had been exceeded, this is indicated in the reply.

d) **priority** – In all modes of propagation, each DSA is responsible for ensuring that the processing of operations is ordered so as to support this service control, if present.

e) **localScope** – The operation is limited to a locally defined scope and each DSA shall not propagate the request outside of this.

f) **scopeOfReferral** – If the DSA returns a referral or partial result to a List or Search operation, then the embedded continuation references shall be within the requested scope.

All other service controls need to be respected, but their use does not require any special consideration in the distributed environment.

15.5.2 Extensions

If a DSA encounters an extended operation in the Name Resolution phase of processing and determines that the operation should be chained to one or more DSAs, it shall include unchanged in the chained operation any extensions present.

NOTE – An Administrative Authority may determine that it is appropriate to return a **serviceError** with problem **unwillingToPerform** if it does not wish to propagate an extension.

If a DSA encounters an extension in the execution phase of processing, two possibilities may arise. If the extension is not critical, the DSA shall ignore the extension. If the extension is critical, the DSA shall return a **serviceError** with problem **unavailableCriticalExtension**. A critical extension to a multiple object operation may result in both results and service errors of this variety. A DSA merging such results and errors shall discard these service errors and employ the **unavailableCriticalExtension** component of **PartialOutcomeQualifier** as described in ITU-T Rec. X.511 | ISO/IEC 9594-3.

15.5.3 Alias dereferencing

Alias dereferencing is the process of creating a new target object name, by replacing the alias entry distinguished name part of the original target object name with the **AliasedEntryName** attribute value from the alias entry. The **object** name in the operation is not affected by alias dereferencing.

15.5.4 Resolving context-variant names

During the name resolution phase, as RDNs are processed, a new target object name is created by ensuring that every **AttributeTypeAndDistinguishedValue** in the RDN uses the primary distinguished value of that attribute as its **value**. In this way, the target object name is progressed towards a primary distinguished name. This is done to provide consistent name handling, in particular where pre-1997 DSAs may be involved in name resolution. The **object** name in the operation is not affected by this substitution.

15.5.5 DirQOP and Protection Request

The **dirqop** selected on the DSP Bind and the **ProtectionRequest** field in the DAP request govern the operation of various security features associated with distributed directory operation.

The selection of point-to-point protection for DSP requests and responses (including chaining arguments and results) is indicated by the **dirqop** established between the DSAs on DSP Bind.

The selection of protection preferred by the requestor for chained DAP Results or Errors is indicated by the **SecurityParameters.ProtectionRequest** field in the DAP request or, if this field is not present, the **SecurityParameters.ProtectionRequest** field in the **ChainingArguments** is to be set to reflect the **dirqop** in the DAP bind.

Where a DAP request is signed, the protection is maintained in the chained DAP request. Any encryption on a DAP request is removed before chaining, although this information may be protected using point-to-point protection of the DSP requests, if required. Encryption of errors is also applied only on a point-to-point basis.

The Security Parameters govern the distributed operation of various security features associated with a distributed directory operation.

15.6 Authentication of Distributed Operations

Users of the Directory together with Administrative Authorities that provide directory services may, at their discretion, require that directory operations be authenticated. For any particular directory operation, the nature of the authentication process will depend upon the security policy in force.

Two sets of authentication procedures are available which collectively enable a range of authentication requirements to be met. One set of procedures are those provided by Bind: these facilitate authentication between two directory application-entities for the purposes of establishing an association. The Bind procedures accommodate a range of authentication exchanges from a simple exchange of identities to strong authentication.

In addition to the peer entity authentication of an association as provided by Bind, additional procedures are defined within the directory to enable individual operations to be authenticated. Two distinct sets of directory authentication procedures are defined. One facilitates originator authentication services, which address the authentication, by a DSA, of the initiator of the original service request. The second set facilitates results authentication services which address the authentication, by an initiator, of any results that are returned.

For originator authentication, two procedures are defined, one based upon a simple exchange of identities, termed **identity based authentication**, and one based upon digital signature techniques, termed **signature based authentication**. The former of these procedures is rudimentary in nature since the identity exchange is based upon the exchange of distinguished names which are transmitted in the clear.

For authentication of results a single **results authentication** procedure is defined, based upon digital signature techniques; due to the generally complex nature of results collation, a simpler, identity-based procedure is not defined.

Authentication of error responses may be supported by these procedures.

The services described below are to be considered as augmenting those provided by the Bind service; Bind procedures are assumed to have been effected successfully prior to authentication of directory operations.

The procedures to be effected by a DSA in providing originator and results authentication are specified in clause 22.

16 The Operation Dispatcher

The **Operation Dispatcher** is the main controlling procedure in a DSA. It guides each operation through the three phases of processing a request. The **Operation Dispatcher** therefore makes use of a set of procedures to fully process the request as shown in Figure 6.

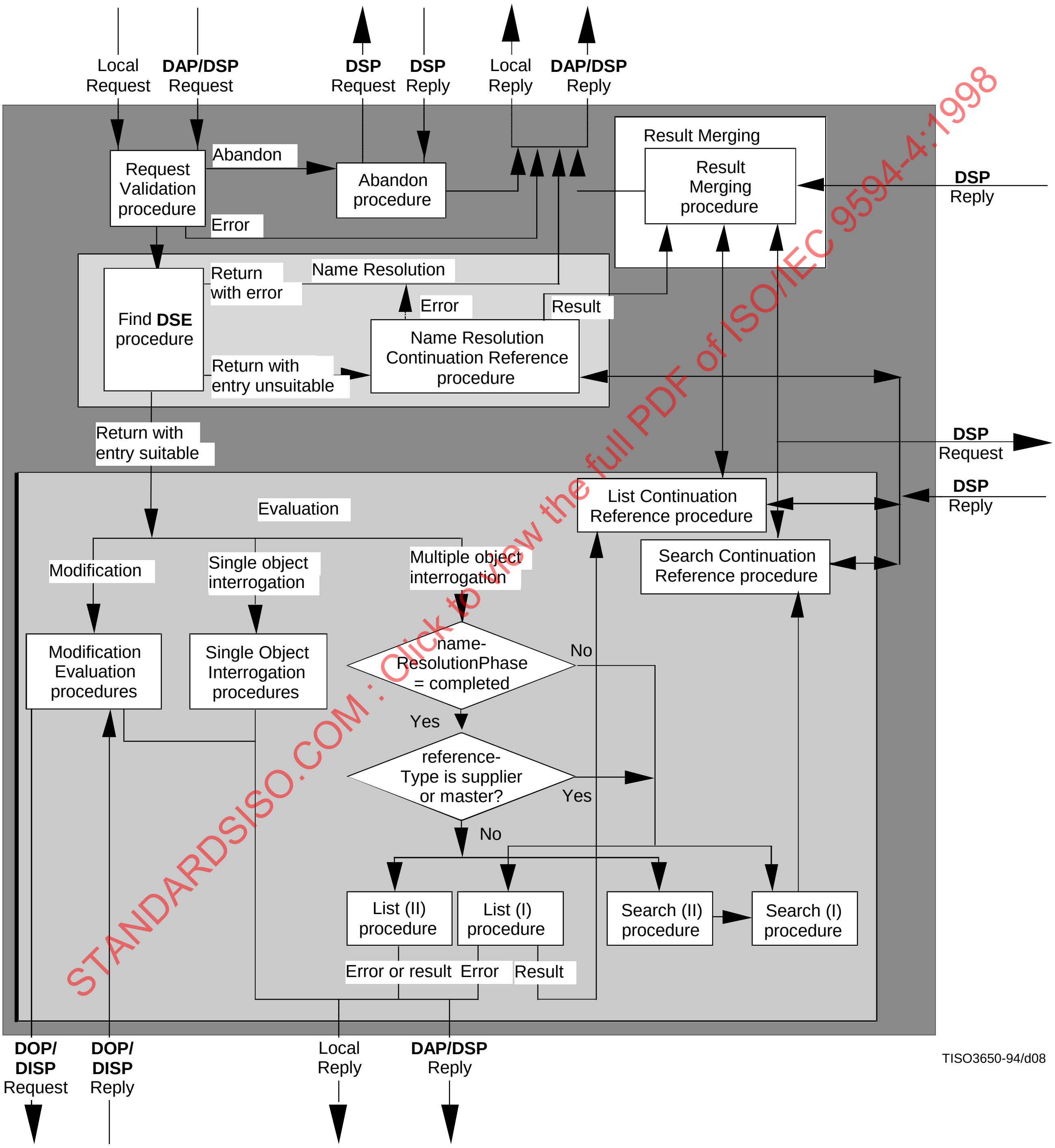


Figure 6 – Operation Dispatcher

16.1 General Concepts

16.1.1 Procedures

Each of the procedures employed by the operation dispatcher consists of a definition of its conceptual interface in terms of its parameters, i.e. arguments, results and errors, and a description of the procedure steps itself. The behaviour of the procedures is described by flowcharts and text. Within a flow chart, the used symbols have the following semantics (see Figure 7):

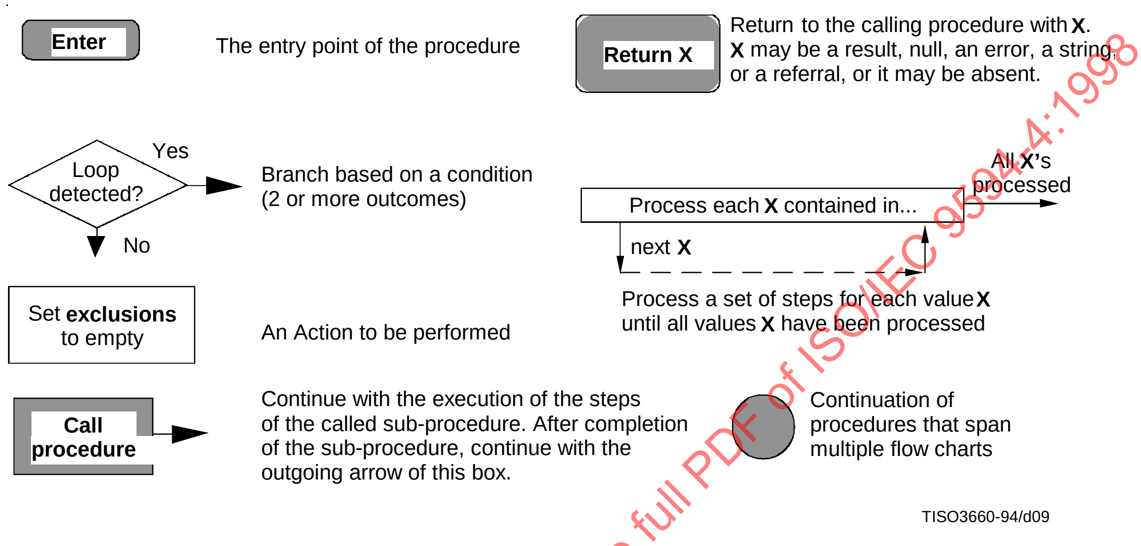


Figure 7 – Symbols Used in Flow Charts

16.1.2 Use of common data structures

All procedures make use of some data structures that are available during the processing of an operation within the **Operation Dispatcher**. These data structures serve to coordinate the data flow within the **Operation Dispatcher**. Most of these structures are directly associated with the argument of the operation and the result to be created for the operation. Components of the argument and result are referred to using their names within the associated ASN.1 definition (e.g. the **operationProgress** component of the chaining arguments). If any of these structures is a compound structure, a component of this structure may be referred to as **compound.component** (e.g. **operationProgress.nameResolutionPhase**).

The following data structures are defined within the **Operation Dispatcher**:

– **NRcontinuationList** – A list of continuation references created for use in the **Name Resolution Continuation Reference** procedure.

– **SRcontinuationList** – A list of continuation references created for use in the **List** or **Search Continuation Reference** procedure.

– **admPoints** – A list of references to DSEs of type administrative point that is collected during Name Resolution.

– **referralRequests –** A list of the requests or subrequests which have been chained as a result of executing referrals. Each such request/subrequest is summarised in the form of a **TraceItem**. This list is used by the Loop Avoidance procedure of 15.4.2.

Further, a procedure may use a set of locally defined variables.

16.1.3 Errors

At each stage of the processing an error may be detected during the execution of any sub-procedure. The error identified within this sub-procedure is normally returned to the requester as a corresponding protocol error. In this case, the operation dispatcher is terminated immediately. In the case that multiple errors are received, local procedures may select one of them to be returned.

Alternatively, a procedure may choose to process errors (e.g. if a **serviceError** with problem **busy** is returned to a chained search subrequest) at certain points of operation processing. In this case, the procedure continues with its execution and no error is returned to the requester.

The DSA may optionally sign, encrypt, or sign and encrypt the errors returned in a distributed operation based on the selected DirQOP and error protection requested.

16.1.4 Asynchronous events

During the processing of an operation request within the Operation Dispatcher several asynchronous events may occur. The following paragraphs specify how to handle an exceeded time limit or size limit or administrative limit, a loss of association and an Abandon request for an operation that is being processed. The handling of all other asynchronous events, e.g. local policy decisions etc., is outside the scope of this Directory Specification.

16.1.4.1 Time limit

A **timeLimit** as specified in the **CommonArguments** can expire at any point in time during the operation. In this case, normally a **serviceError** with problem **timeLimitExceeded** is returned to the requesting DUA or DSA and the operation dispatcher is terminated. Alternatively, a procedure may choose to handle this event in a different way (e.g. during processing of a **search** request).

If a DSA receives a request from another DSA with the time limit exceeded, it shall send a **serviceError** with problem **timeLimitExceeded** without any further processing of the request.

If a DSA has outstanding (sub)requests, when the **timeLimit** expires, and there are no results available, it shall return a **serviceError** with problem **timeLimitExceeded** to the requester.

If a DSA has outstanding subrequests, when the **timeLimit** expires, and there are results available, it shall return a result to the requester with the following contents:

a) all the collected results, up to the **timeLimit** expiring;

b) the **limitProblem** component of the **partialOutcomeQualifier** result-parameter shall be set to **timeLimitExceeded**;

c) the **unexplored** component of the **partialOutcomeQualifier** result-parameter shall contain a continuation reference value for each set of DSAs to which subrequests were sent but the result of which are not included in the result to the requester, in addition to continuation references to DSAs to which this DSA did not attempt to send subrequests.

16.1.4.2 Loss of an association

If the association to the requester is lost, the DSA may optionally for each outstanding interrogation (sub)request, send an **chainedAbandon** request, unless the association to the DSA in question has also been lost. All replies to such **chainedAbandon** requests and all replies to outstanding (sub)requests shall be discarded.

If the association to one of the outstanding chained subrequests is lost and the association with the requester is not lost, the DSA may, for interrogation operations only, optionally try any alternative reference to another DSA that is able to process the chained request (e.g. a reference to a shadow DSA, after loss of the association to the master DSA). If this does not succeed, the DSA shall act as follows:

1) If **operationProgress.nameResolution** is set to **notStarted** or **proceeding**, return either a **serviceError** with problem **unavailable** to the requester or a referral error whose continuation reference contains the set of DSAs that are able to continue the operation. If non-specific subordinate references are used during the Name Resolution phase and not all the associations in question are lost, optionally attempt to do the name resolution without the DSAs to which the associations are lost. If this fails, return either a **serviceError** with problem **unavailable**, or a referral error containing the complete set of NSSRs.

   NOTE – It is a local choice which type of error is returned.

2) If **operationProgress.nameResolution** is set to **complete** and the request is a single object operation, return a **serviceError** with problem **unavailable** to the requester.

3) If **operationProgress.nameResolution** is set to **complete** and the request is a multiple entry interrogation operation, the DSA shall add a continuation reference to **partialOutcomeQualifier.unexplored** of the operation result, with **AccessPointInformation** identifying the set of DSAs that are able to continue the operation, including any DSAs to which associations have been lost.

16.1.4.3 Abandoning the operation

During the processing of an operation, an Abandon request can be received for this operation. In this case, during the processing of the Abandon request, the **Abandon** procedure is called for the operation to be abandoned.

16.1.4.4 Administrative Limits

There may be limits imposed by the local DSA administrator, e.g. the amount of time to spend on processing a request, or the maximum size of data to be returned. If any of these limits are exceeded the DSA shall return **serviceError** with problem **administrativeLimitExceeded**.

16.1.4.5 Size Limit

A size limit, as specified in **CommonArguments**, can be exceeded at any point in time during processing of a List or Search operation. In this case, a partial result (taken from the set of already collected results) shall be returned to the requester with **limitProblem** set to **sizeLimitExceeded**. In addition, the **unexplored** component may be used for returning Continuation References of unaccessed DSAs. Operation Dispatcher is then terminated.

16.2 Procedures of the operation dispatcher

The procedure that is performed by the **Operation Dispatcher** for processing each received request (either over DAP or DSP) is defined by the following steps. Due to alias dereferencing, this procedure may also call itself (a local request), in which case a local reply (rather than a DAP or DSP reply) is returned.

1) Validate several aspects of the operation arguments (**Request Validation** procedure). If an error is encountered during validation, return this error locally or over DAP/DSP.

2) If the operation received was an Abandon operation, call the **Abandon** procedure and return a reply afterwards.

3) Resolve the name of the target object by executing the **Find DSE** procedure (which includes the **Target Found** and **Target Not Found** sub-procedures). If the requested entry was found and is suitable (according to the setting of the service controls, chaining arguments and local policy decisions), continue with the **Evaluation Phase** at step 6). If during Name Resolution an error was encountered, it is returned. If the entry was found not to be suitable, continue at step 4).

4) The **Name Resolution Continuation Reference** procedure is called to process the list of Continuation References as stored in the **NRcontinuationList**. In order to process these Continuation References, chained requests may be issued to other DSAs (if service controls and local policy decision allow it).

   In case of an error, this error is directly returned either locally or via DAP/DSP. If the chained request generated a result, then continue with step 5).

5) The **Result Merging** procedure is called to merge the local results with the received Chained Results. If the Chained Results contain embedded Continuation References, these may first be resolved if the service controls and local policy allow or require it.

   This may cause additional Chained Requests to be issued (whose Chained Results may also contain embedded Continuation References).

   The merged results are returned to the caller, and processing of the request ceases.

   If protection is performed on the results, the merging of results shall not be performed.

6) If the operation is a modification operation, continue at step 7).

   If the operation is a single entry interrogation operation, continue at step 8).

   If the operation is a multiple entry interrogation operation, continue at step 9).

7) When carrying out a modification procedure, Operational Bindings may need to be established, modified or terminated, or shadows may need to be updated as a consequence of performing the operation. Whether these are done synchronously or asynchronously with the performance of the original operation depends on the respective modification operations (and on local policy). A local or a DAP/DSP result or error is returned to the caller.

8) The result of a single entry interrogation operation is directly returned to the caller as a local or a DAP/DSP result.

9) If the operation is a multiple entry interrogation operation, then check the **nameResolutionPhase** of the operation. If it is not **completed**, then call the **List(I)** or **Search(I)** procedure, otherwise call the **List(II)** or **Search(II)** procedure, respectively.

10) The outcome of a call to the **List(II)** procedure (result or error) and the outcome of a call to the **List(I)** procedure (in case that the outcome is an error) can directly be returned to the caller (as a local or a DAP/DSP result).

    If the procedure called was the **List(I)** procedure, the result might contain Continuation References that have to be dereferenced (depending on service controls and local policy). This may result in chained List operations being sent off to the respective DSAs. To merge the results continue at step 5) with the call to the **Result Merging** procedure.

11) If the operation was a Search operation, any Continuation References are resolved by the **Search Continuation Reference** procedure (if required and allowed). This may cause Chained Search requests to be sent off to the respective DSAs. The **Result Merging** procedure [see step 5)] is called to merge the search results and to possibly dereference contained Continuation Reference, if any.

16.3 Overview of procedures

This clause gives an overview of the basic functionality of the procedures employed by the **Operation Dispatcher** which are defined in clauses 17 through 22.

16.3.1 Request Validation procedure

This procedure, described in clause 17, is called to perform loop checking, limit checking, and security checking prior to performing local name resolution. This procedure also provides default settings for those parameters of the **ChainingArgument** that are not provided by the DAP in the case that the request came from a DUA. Further, this procedure singles out any **abandon** request and notifies this to **Operation Dispatcher**.

16.3.2 Abandon procedure

This procedure, described in 20.5, tries to find the operation that is to be abandoned and terminate it. If there are any outstanding subrequests, Chained Abandon operations may be sent after them. The procedure either returns an empty result to the caller, or an error indication (e.g. **abandonError** with problem **tooLate**).

16.3.3 Find DSE procedure

This procedure, described in 18.2 and 18.3, matches the components of the name of the target object against the locally held DSEs to resolve the target object name. If an alias DSE is encountered, the alias is dereferenced (if permitted) and the procedure is restarted to resolve the new name.

If the target was not found, the procedure is continued at the **Target Not Found** sub-procedure. If the target was found, the procedure is continued at the **Target Found** sub-procedure.

NOTE – **Target Not Found** and **Target Found** are continuations of the **Find DSE** procedure.

The procedure may result in various errors, in which case, the associated protocol error is returned to the requester and the **Operation Dispatcher** is terminated.

16.3.3.1 Target Not Found sub-procedure

This procedure, described in 18.3.2, performs an evaluation of the located intermediate DSEs and creates a set of Continuation References in **NRcontinuationList,** based on the set of knowledge references that have been detected during the **Find DSE** procedure. This set of references is then further processed within the **Name Resolution Continuation Reference** procedure.

The procedure may result in various errors, in which case the associated error is returned to the requester and the **Operation Dispatcher** is terminated.

16.3.3.2 Target Found sub-procedure

This procedure, defined in 18.3.3, checks if the found DSE is suitable for the requested operation, i.e. in the case where it is shadowed information. This may include checking the suitability of the whole subtree of shadowed information below the target object in the case of a multiple object operation (e.g. subtree search).

If the located entry is suitable, the appropriate operation evaluation procedure is invoked. Otherwise, a **ContinuationReference** pointing to the supplier (or master) of the information is created in **NRcontinuationList** and the **Name Resolution Continuation Reference** procedure is invoked.

16.3.4 Single entry interrogation procedure

This procedure, described in 19.2, is invoked to actually execute those operations that only affect a single entry, i.e. Read and Compare operations. After completion, a reply (result or error) created by the procedure is returned to the requesting DSA/DUA.

16.3.5 Modification procedures

These procedures, described in 19.1, are executed to process the modification operations, i.e. Add Entry, Remove Entry**,** Modify Entry and Modify DN. This is done by executing a specific sub-procedure defined for each of these operations. During (or after) these sub-procedures, DOP and DISP requests may be issued to other DSAs. After successful completion, a result (created by the sub-procedures) is returned to the requesting DSA/DUA.

16.3.6 Multiple entry interrogation procedures

These procedures, described in 19.3, are executed to process operations that affect multiple entries which may or may not be located in the same DSA. This is done by executing specific sub-procedures defined for each of the Search and List operations to accomplish request decomposition. These procedures create a local result of the operation evaluation and optionally a set of continuation references in **SRcontinuationList**. If **SRcontinuationList** is empty at the end of this procedure, the created result is directly returned to the requesting DSA/DUA. Otherwise, these continuation references are processed by invoking **List** or **Search Continuation Reference** procedure, according to the operation type.

16.3.7 Name Resolution Continuation Reference procedure

This procedure, described in 20.4.1, processes the continuation references in **NRcontinuationList** created during the Name Resolution phase. These continuation references are either used to issue chained subrequests or returned in a referral error. In the case of chaining, the results or errors returned from the chained request are returned for further processing by the **Result Merging** Procedure.

16.3.8 List and Search Continuation Reference procedure

These procedures, described in 20.4.3 and 20.4.4, process the continuation references in **SRcontinuationList** created by the multiple entry interrogation procedures and either resolve them by issuing chained subrequests or by creating continuation reference(s) within the **partialOutcomeQualifier.unexplored**. When results or errors for all outstanding subrequests have been received, they are returned for further processing by the **Result Merging** Procedure.

16.3.9 Result Merging procedure

This procedure, described in clause 21, either examines the result from a chained request or combines the local operation results with the results received from the chained subrequests. If a subrequest had returned an error, this procedure determines how this error has to be handled.

If there are any continuation references left in the result, they will (if local policy allows so and service controls require it) be dereferenced by the **Name Resolution**, **List**, or **Search Continuation Reference** procedures, accordingly. Duplicates are removed from the result if it is unsigned.

The merged result (with all merged results and unresolved continuation references) is returned to the requesting DUA/DSA.

If protection is performed on the results, the merging of results shall not be performed.

17 Request Validation procedure

17.1 Introduction

The **Request Validation** procedure is the entry point of the Operation Dispatcher for inputs from DUAs and DSAs, preparing such inputs for Name Resolution processing. The function of this procedure is to detect abandon operations, to perform security checks, to adjust input received from DUAs so that it may be processed in the same way as input received from DSAs, to check the arguments of the request for valid syntax and semantics, to perform loop detection, and to perform other miscellaneous checks. The flow of **Request Validation** is depicted in Figure 8.

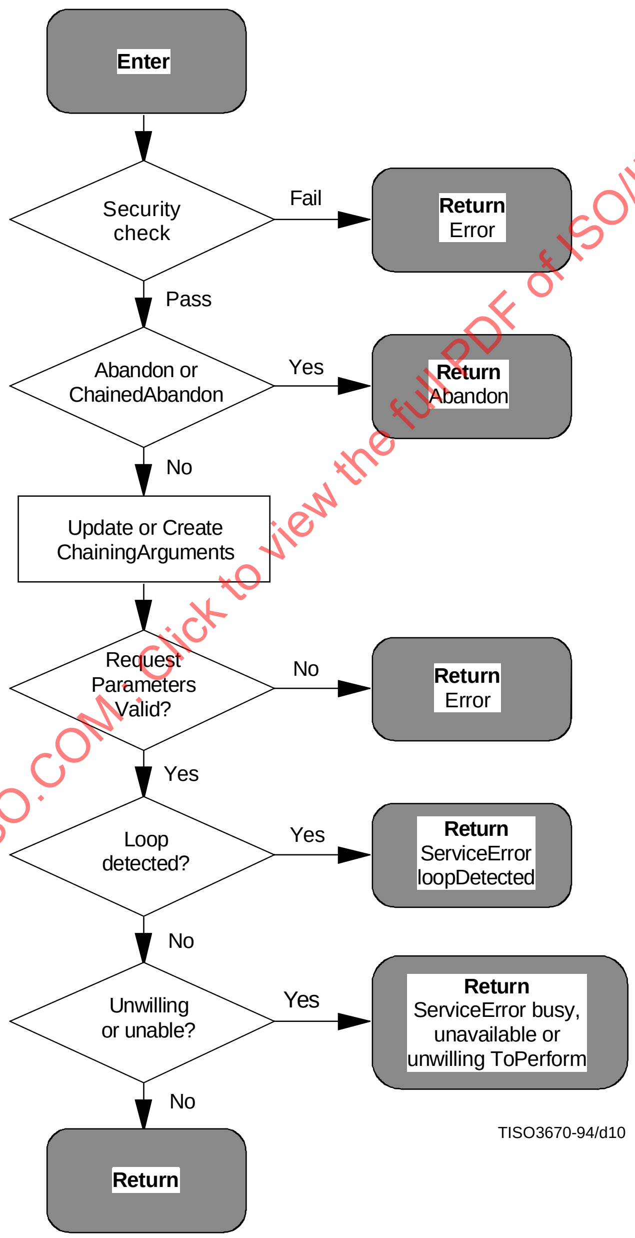


Figure 8 – Request Validation procedure

17.2 Procedure parameters

17.2.1 Arguments

The input argument to **Request Validation** consists of **ChainingArguments** (except in the case of **chainedAbandon** operations), if the request is received from a DSA, and the argument issued by the originator of the request.

17.2.2 Results

The output result of **Request Validation** consists of five possibilities.

a) If the security check fails, an error is returned to the requester.

b) If the input is an **abandon** or **chainedAbandon** operation, the output is the argument of the operation.

c) If the arguments of the request are invalid, then an error is returned to the requester. Depending on local policy, the DSA may choose whether to return a **serviceError** or a **securityError**.

d) If a loop is detected, a **serviceError** with problem **loopDetected** is returned to the requester.

e) If, based on resource problems or policy considerations, the DSA is unable or unwilling to perform the operation, a **serviceError** (with problem **busy**, **unavailable**, or **unwillingToPerform**) is returned to the requester.

f) In all other cases, the validated input, transformed by addition of **ChainingArguments** if received from a DUA or the update of **ChainingArguments.traceInformation** if received from a DSA, is the output of the procedure and subsequently the input to the **Name Resolution** procedure.

17.3 Procedure definition

The security check described in 17.3.2 is performed. This may result in the return of an error and the termination of the Operation Dispatcher.

If the input is an **abandon** or **chainedAbandon** operation, only the steps in 17.3.1 are subsequently performed, otherwise the steps in 17.3.3-17.3.5 are performed. 17.3.5 describes the loop detection procedure which may result in the return of an error and the termination of the Operation Dispatcher.

Next the checks in 17.3.6 are performed. They may result in the return of an error and the termination of the Operation Dispatcher.

If the checks in 17.3.2-17.3.6 do not result in the termination of the Operation Dispatcher, the steps in 17.3.7 are performed and the procedure terminates with the transfer of its output to the **Name Resolution** procedure.

17.3.1 Abandon processing

The argument of an **abandon** or **chainedAbandon** is passed to the **Abandon** procedure, (see 20.5), to process the abandon request.

17.3.2 Security checks

If the argument to the operation is signed, encrypted, or signed and encrypted, the signature may be checked. Should the signature be invalid or the decryption fail, or be absent in a case when it should be present, an error may be returned to the requestor. Alternatively, a DSA may perform any other locally defined action.

17.3.3 Input preparation

17.3.3.1 DUA request

If the operation is received from a DUA, a **ChainingArguments** value is created as follows:

a) **ChainingArguments.originator** is set as described in 10.3.

b) **ChainingArguments.operationProgress** is set to the value of **CommonArguments.operationProgress**.

c) **ChainingArguments.traceInformation** is set to a sequence containing a single **TraceItem** value. This value is constructed as follows. **TraceItem.dsa** is set to the name of the DSA executing **Request Validation**. **TraceItem.targetObject** shall be omitted. **TraceItem.operationProgress** is set to the incoming value.

d) If the service control of the operation specifies a time limit (the available elapsed time in seconds for completion of the operation), **ChainingArguments.timeLimit** is set to the (UTC) time by which the operation shall complete to meet the user's specified time limit.

e) **ChainingArguments.AuthenticationLevel** and **ChaingingArguments.UniqueIdentifier** are set according to the local security policy.

f) If the **manageDSAIT** option is set in the **ServiceControls**, then:

   – the **nameResolutionPhase** component of **operationProgress** shall be set to **completed**;

   – the **nextRDNToBeResolved** component of the **operationProgress** shall be omitted;

   – **referenceType** shall take the value **self**;

   – **entryOnly** shall take the value **FALSE**;

   – **nameResolveOnMaster** shall take the value **FALSE**; and

   – the **chainingProhibited** option in **ServiceControls** shall be set;

   – the remaining optional elements of **ChainingArguments** are omitted, with default values being assumed where specified.

g) If the **manageDSAIT** option is not set in the **ServiceControls**, then the remaining optional elements of **ChainingArguments** are omitted, with default values being assumed where specified.

h) **ChainingArguments.SecurityParameters.ProtectionRequest** is used to indicate the level of protection (sign, encrypt, or sign and encrypt) to be applied to the results.

17.3.3.2 DSA request

If the operation is received from a DSA, **ChainingArguments.traceInformation** is updated by appending a value at the end of sequence **TraceItem**. This value is constructed as follows:

a) **TraceItem.dsa** is set to the name of the DSA executing **Request Validation**.

b) **TraceItem.targetObject** is set to the value of **ChainingArguments.targetObject** unless the **object** (or **baseObject** in the case of a Search operation) of the request argument is identical to **ChainingArguments.targetObject**, in which case **TraceItem.targetObject** shall be omitted.

c) **TraceItem.operationProgress** is set to the value of **ChainingArguments.operationProgress**.

17.3.4 Validity assertion

The operation shall be checked for valid syntax and semantics of its arguments according to the rules contained in the clauses defining each operation (e.g. it should be checked that the **nextRDNToBeResolved** does not provide a number exceeding the number of RDNs in the **targetObject**). If the request is detected to contain invalid arguments, the operation is terminated and an error is returned to the user, depending on the kind of invalidity detected.

17.3.5 Loop detection

If any two **TraceItem** values of **ChainingArguments.traceInformation** (as prepared in 17.3.3) are identical, processing of the operation has returned to a previous state, i.e. a loop has been detected. In this case, a **serviceError** (with problem **loopDetected**) shall be returned to the requester and the Operation Dispatcher terminates.

17.3.6 Unable or unwilling to perform

Request Validation may assess available resources and determine that the operation cannot be performed. It may also determine, based on policy considerations, that the operation should not be performed. In these cases, a **serviceError** (with problem **busy**, **unavailable**, or **unwillingToPerform**) may be returned to the requester and the Operation Dispatcher terminates.

17.3.7 Output processing

In the final phase of **Request Validation** the validated input, transformed by addition of **ChainingArguments** if received from a DUA or the update of **ChainingArguments.traceInformation** if received from a DSA, is returned and employed as input to the **Name Resolution** procedure.

18 Name Resolution procedure

18.1 Introduction

This clause describes the **Name Resolution** procedure, its Arguments, Results, and its possible Error conditions. As shown in Figure 6 (**Operation Dispatcher**), the **Name Resolution** procedure consists of two procedures:

– **Find DSE** procedure;

– **Name Resolution Continuation Reference** procedure.

The **Find DSE** procedure is described in three flow charts, namely **Find DSE**, **Target Found**, and **Target Not Found**. The **Find DSE** procedure matches the target entry name to locally stored DSEs, component by component. If the target entry is found locally, then **Find DSE** continues with the **Target Found** sub-procedure, which then calls the **Check Suitability** procedure to check the suitability of the found DSE for evaluation. If the target entry is not found locally, then **Find DSE** continues with the **Target Not Found** sub-procedure prepares Continuation Reference(s) to be added to the **NRcontinuationList** for the **Name Resolution Continuation Reference** procedure to dispatch it.

NOTE 1 – **Name Resolution** shall perform name matching against multiple distinguished values differentiated by context, as described in 9.4 of ITU-T Rec. X.501 | ISO/IEC 9594-2, when determining a match.

NOTE 2 – **Name Resolution** may fail if a pre-1997 superior DSA holds a subordinate reference to an entry held in a later edition DSA and the RDN for that entry includes contexts. **Name Resolution** will fail against the shadow copy of an entry when an alternative name is used as a purported name and the shadow entry is held in a first or second edition DSA.

18.2 Find DSE procedure parameters

18.2.1 Arguments

The procedure uses the following arguments:

a) **ChainingArguments.aliasDereferenced**;

b) **ChainingArguments.aliasedRDNs**;

c) **ChainingArguments.excludeShadows**;

d) **ChainingArguments.nameResolveOnMaster**;

e) **ChainingArguments.operationProgress (nameResolutionPhase, nextRDNToBeResolved)**;

f) **ChainingArguments.referenceType**;

g) **ChainingArguments.targetObject**;

h) the operation type;

i) the operation argument;

NOTE – Where no actual values exist, default or implied values are used, as specified in 10.3.

18.2.2 Results

There are two cases of successful outcome from **Find DSE** (indicated by **entry suitable** or **entry unsuitable**):

The first successful case returns (from the **Target Not Found** sub-procedure) Continuation Reference(s) in **NRcontinuationList** which is then passed on to the **Name Resolution Continuation Reference** procedure to continue the Name Resolution phase.

The second successful case returns (from the **Target Found** sub-procedure) a (reference to a) DSE, which is passed to one of the Evaluation procedures.

18.2.3 Errors

The following errors may be returned:

a) **serviceError: unableToProceed, invalidReference, unavailableCriticalExtension**;

b) **nameError: noSuchObject, aliasDereferencingProblem, contextProblem**.

18.2.4 Global Variables

The procedure uses the following global variable:

– **NRcontinuationList** list to store the Continuation Reference(s) needed to continue name resolution in the **Name Resolution Continuation Reference** procedure.

18.2.5 Local and Shared Variables

The procedure uses the following local variables:

a) **i** Index used to identify the component of the target name being worked on.

b) **m** The length of the target object name to be used in name resolution. For operations that name resolve to the parent entry, i.e. Add Entry, **m** is set to (the number of RDNs in the target object) – 1. For all other operations, **m** is set to the number of RDNs in the target object.

c) **lastEntryFound** Index, so that DSE(lastEntryFound) is the last matched DSE that is of type **entry**.

d) **lastCP** Index, so that DSE(lastCP) is the last shadowed context prefix encountered.

e) **candidateRefs** A set of continuation references.

The shared variable **admPoints** (defined in Operation Dispatcher) is also used. For convenience, component **i** of the target object name is denoted as **N(i)**.

18.3 Procedures

NOTE – There are some texts in the flow chart that are only relevant to specific operations. This is not shown in the flow charts, but is described in the accompanying text.

18.3.1 Find DSE procedure

See Figure 9.

This procedure attempts to resolve the target object name locally.

1) Initialize the local variables **lastEntryFound** and **lastCP** to **0**; **admPoints** and **candidateRefs** to an empty set, and initialize **i** to **0**.

2) Compare **i** and **m**. If they are not equal, then continue at step 5).

3) If they are equal, check if **nameResolutionPhase** is **completed**. If not **completed**, continue at **Target Not Found** sub-procedure.

   If the **nameResolutionPhase** is **completed** and the **manageDSAIT** critical extension is set, then return with **entry suitable**.

4) If **nameResolutionPhase** is **completed**, then check if any immediate subordinate of DSE(i) is a context prefix (of type **cp**).

   If one (or more) immediate subordinate DSE(s) is of type **cp**, then return with **entry suitable**.

   NOTE 1 – This case is for **List (II)** and **Search (II)** subrequests.

   If no immediate subordinates of DSE(i) are of type **cp**, then continue at **Target Not Found** sub-procedure.

5) Try to find a match for the (**i + 1**)-th component of the target object name with the name of a subordinate of the last matched DSE. In the case of i = 0, try to match one of the DSEs immediately subordinate to the root DSE. If no match can be found, continue at **Target Not Found** sub-procedure. If a single match is found, increment **i**, and store the matched DSE as the **i**-th element in the vector of found DSEs.

   NOTE 2 – Name matching includes handling of multiple distinguished values differentiated by context, where known, as described in 9.4 of ITU-T Rec. X.501 | ISO/IEC 9594-2.

   If more than one match is found, then return a **nameError** with problem **contextProblem**.

   NOTE 3 – For example, this can be the case when an **AttributeTypeAndDistinguishedValue** in a purported name contains multiple distinguished attribute values differentiated by contexts and different of these values match values in different target names.

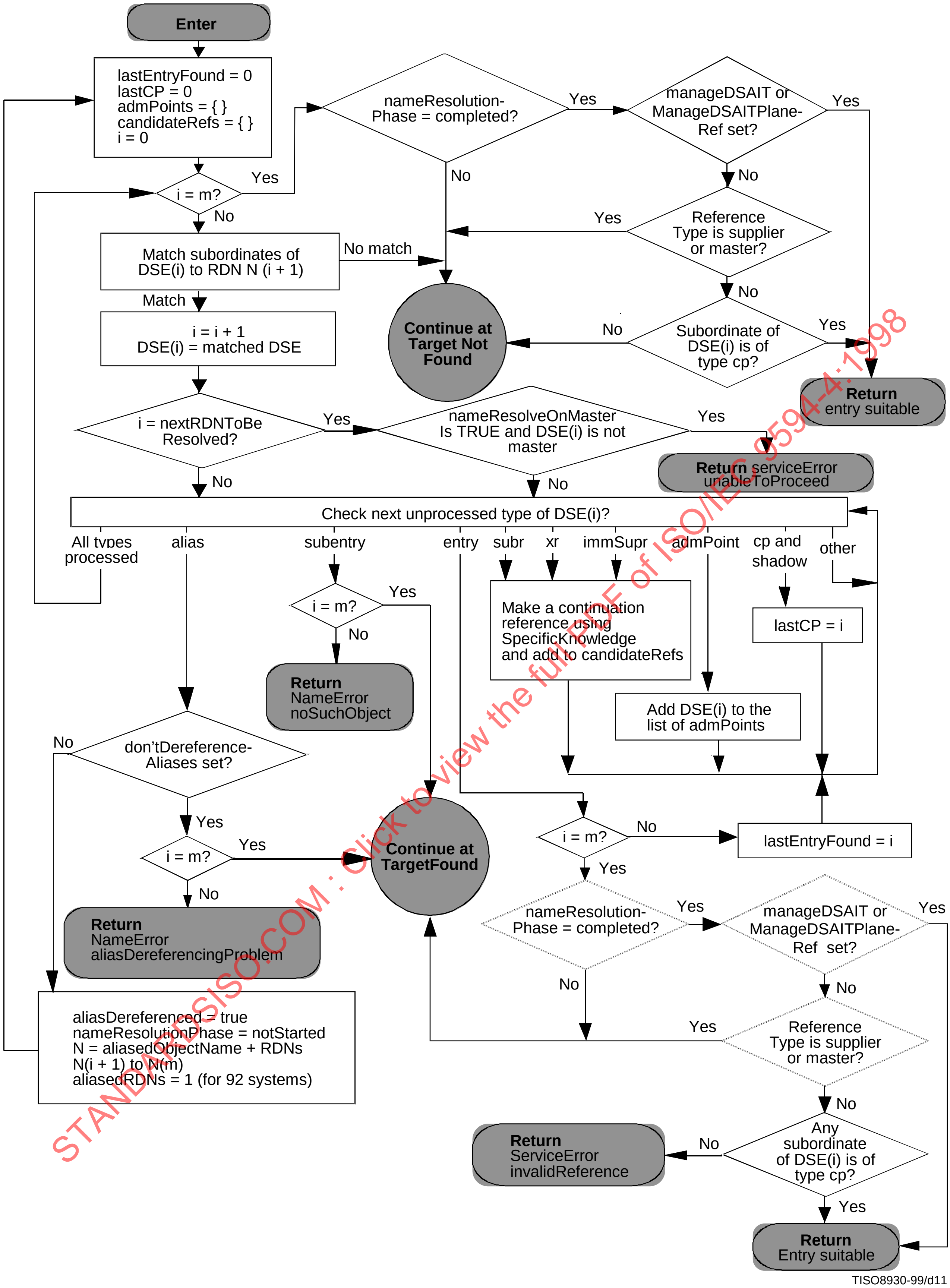


Figure 9 – Find DSE procedure

6) If **i** equals **nextRDNToBeResolved**, then check if the following two conditions are both met:

   – the **ChainingArgument.nameResolveOnMaster** is **TRUE**;

   – DSE(**i**) is not a master entry.

   If both conditions are met, then return **serviceError** with problem **unableToProceed.**

   NOTE 4 – This indicates the use of **nameResolveOnMaster** to avoid multiple paths to the same target object.

7) Check all the DSE type bits of DSE(**i**). For each type bit, some processing is potentially required. The action to take for each type found is given below:

   – If both the **cp** and **shadow** bits are set, then remember the index **i** in **lastCP**.

   – If the **admPoint** bit is set, check the **administrativeRole** operational attribute. If this is the start of an autonomous administrative area then empty the **admPoints** list. If this is the start of one or more specific administrative areas, then check the **admPoints** list and remove any existing points that are no longer relevant (i.e. their roles have been superseded by the new administrative point). Store DSE (**i**) in the list.

   – If one of the **subr**, **xr** or **immSupr** bits is set, then generate a continuation reference using the **specificKnowledge** attribute with **operationProgress.nameResolutionPhase** set to **proceeding**, **nextRDNToBeResolved** set to **i**, **targetObject** constructed from the resolved components using primary RDNs (alternative distinguished values may be included in the RDNs) concatenated with the remaining unresolved components, and **accessPoints** and **referenceType** set as appropriate. Add the continuation reference to the list of continuation references in **candidateRefs**.

   – If the **entry** bit is set, then test for **i** equal to **m** (and therefore the target object name being completely matched). If **i** does not equal **m**, then remember the found entry by setting **lastEntryFound** to **i** and continue processing the type bits of DSE(**i**). If **i** and **m** are equal, continue at step 8).

   – If the **subentry** bit is set, then test for **i** equal to **m** (and therefore the target object name being completely matched). If they are equal, then continue at **Target Found** procedure; if they are not equal, then return a **nameError** with problem **noSuchObject**.

   – If the **alias** bit is set, test if **dontDereferenceAliases** is set.

     If **dontDereferenceAliases** is not set, the alias can be dereferenced. Therefore, set **chainingArguments.aliasDereferenced** to **TRUE**, **nameResolutionPhase** to **notStarted**, the name of the target object to the **aliasedEntryName** as supplied in the alias entry concatenated with the remaining unmatched components of the previous target object name (i.e. concatenate with the (**i** + **1**)-th to **m**-th component of the previous target object name). Post-1988 edition DSAs do not set **aliasedRDNs** (whereas 1988 edition DSAs set **aliasedRDNs** to the number of RDNs in **aliasedEntryName**). Start Name Resolution again by continuing at step 1).

     If **dontDereferenceAliases** is set, then the alias cannot be dereferenced. Check if the target object name has been processed completely by comparing **i** and **m** for equality. If they are equal (and the name therefore fully matched), then continue at **Target Found** sub-procedure. If they are not equal (and the name therefore not fully matched), then return **nameError** with problem **aliasDereferencingProblem**.

   – For all other possible DSE types, no action is needed. Internally mark that DSE type as processed and continue processing the still unprocessed DSE type bits of the DSE(**i**).

   – If all type bits of DSE(**i**) are processed, then continue at step 2).

8) Check if **nameResolutionPhase** is **completed**. If it is not, then continue at **Target Found** sub-procedure.

9) If **nameResolutionPhase** is **completed**, then check if the **referenceType** used is **supplier** or **master**; if so, continue at the **Target Found** sub-procedure.

   NOTE 5 – This is for the chain-to-supplier subrequests.

   If the **nameResolutionPhase** is already **completed** and the **manageDSAIT** critical extension is set, then return with **entry suitable**.

10) Otherwise, check if any of the DSEs immediate subordinate to DSE(**i**) is a Context Prefix (and therefore of type **cp**). If there is (one or more), return **entry suitable**. If none of the immediate subordinate entries is of type Context Prefix, then return a **serviceError** with problem **invalidReference**.

    NOTE 6 – This case is for **List (II)** and **Search (II)** subrequests.

18.3.2 Target Not Found sub-procedure

See Figure 10.

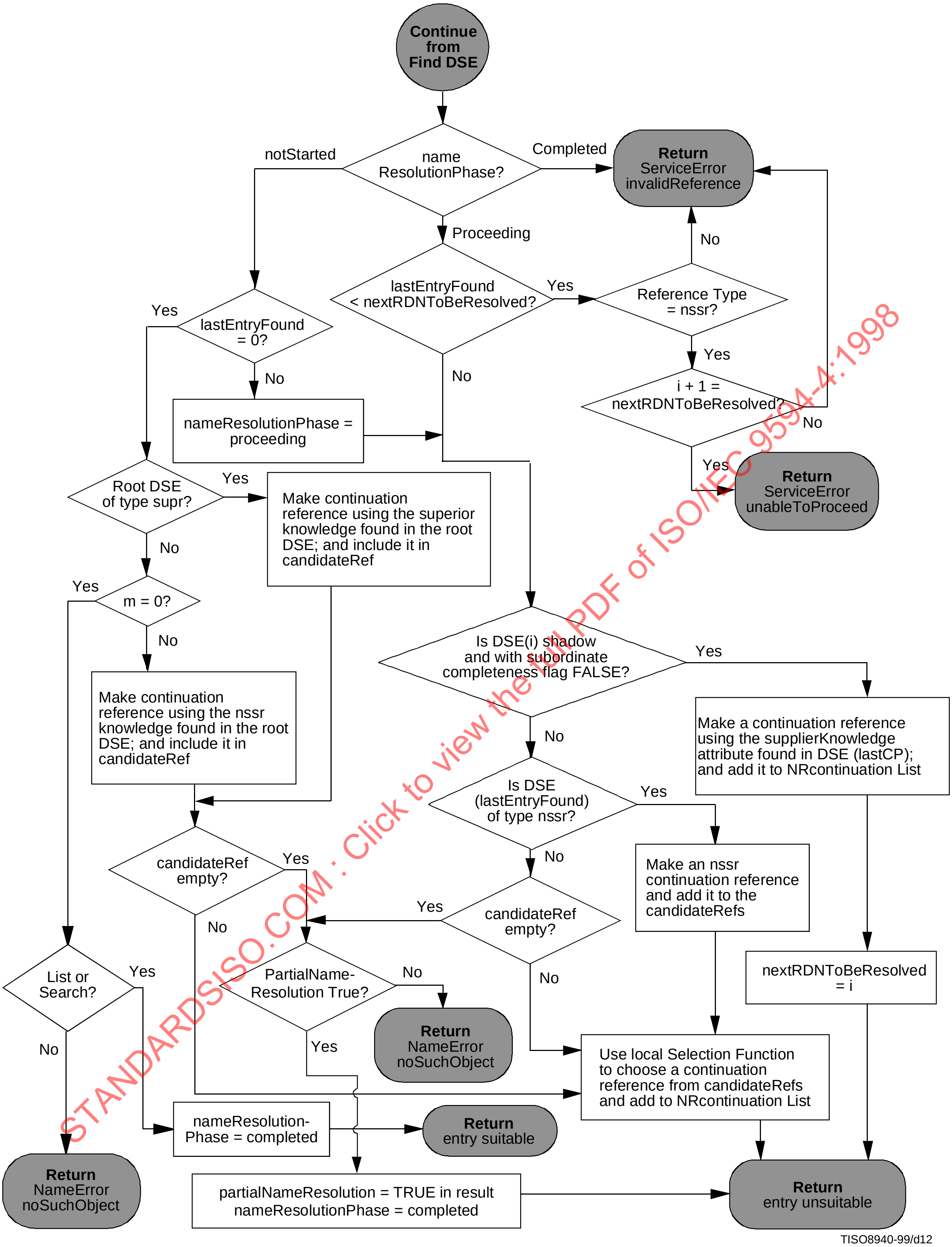


Figure 10 – Target Not Found sub-procedure

This sub-procedure is called when the target object name is not found in the local DSA. This sub-procedure determines the best type of knowledge reference to use to continue name resolution, unless an error is detected in which case the error is returned.

1) When continuing from **Find DSE** procedure, distinguish between the three possible phases of the Name Resolution phase.

   If **nameResolutionPhase** is **notStarted**, continue at step 2).

   If **nameResolutionPhase** is **proceeding**, continue at step 8).

   If **nameResolutionPhase** is **completed**, continue at step 12).

2) If an entry was found (**lastEntryFound** not equal to **0**), set **nameResolutionPhase** to **proceeding** and continue at step 9).

3) If no entry was found (**lastEntryFound=0**), then check if the DSA is a First Level DSA.

   If it is a First Level DSA, then the root DSE does not contain a Superior Reference and therefore is not of type **supr**. In this case, continue at step 4).

   If the DSA is not a First Level DSA, then the root DSE contains a Superior Reference and therefore is of type **supr**. In this case, generate a Continuation Reference using the superior knowledge as found in the root DSE. Set:

   – **targetObject** to the name of the target object constructed from the resolved components using primary RDNs (alternative distinguished values may be included in the RDNs) concatenated with the remaining unresolved components;

   – **operationProgress.nameResolutionPhase** to **notStarted**;

   – **referenceType** to **superior**; and

   – **accessPoints** as appropriate.

   Add the Continuation Reference to the list of Continuation References in **candidateRefs**. Continue at step 6).

4) Check if the operation was directed to the root entry (**m = 0?**). If it was, continue at step 5). If it was not, generate a Continuation Reference using any NSSR knowledge found in the root DSE. Set:

   – **targetObject** to the name of the target object constructed from the resolved components using primary RDNs (alternative distinguished values may be included in the RDNs) concatenated with the remaining unresolved components;

   – **operationProgress.nameResolutionPhase** to **proceeding**;

   – **operationProgress.nextRDNToBeResolved** to **1**;

   – **referenceType** to **nonSpecificSubordinate**; and

   – **accessPoints** as appropriate.

   Add the Continuation Reference to the list of Continuation References in **candidateRefs**. Continue at step 6).

5) At a First Level DSA, only List or Search operations may be performed with the root entry as base object. Therefore, if the operation was not a List or Search operation, return **nameError** with problem **noSuchObject**. If it was a List or Search operation, set **nameResolutionPhase** to **completed** and return with **entry suitable**.

6) Check if there are any Continuation References in **candidateRefs**. If **candidateRefs** is empty and **partialNameResolution** is **FALSE**, return **nameError** with problem **noSuchObject**. If **candidateRefs** is empty and **partialNameResolution** is **TRUE**, then in the result set **partialName** to **TRUE**, **nameResolutionPhase** to **completed**, and return with **entry suitable**. Otherwise, continue at step 7).

7) Use a local selection function to choose a Continuation Reference from the list of Continuation References in **candidateRefs**, add it to the list of Continuation References in **NRcontinuationList** and return with **entry unsuitable**.

8) If the DSA was unable to proceed with Name Resolution (in which case **lastEntryFound** is less than **nextRDNToBeResolved**), continue at step 11). Otherwise, continue with next step.

9) If DSE(**i**) is a shadow DSE with incomplete subordinate knowledge (**subordinateCompletenessFlag** is **FALSE**), then generate a Continuation Reference from the **supplierKnowledge** attribute found in **DSE(lastCP)**. Set:

   – **targetObject** to the name of the target object constructed from the resolved components using primary RDNs (alternative distinguished values may be included in the RDNs) concatenated with the remaining unresolved components;

- **operationProgress.nameResolutionPhase** to **proceeding**;
- **operationProgress.nextRDNToBeResolved** to **lastEntryFound**;
- **referenceType** to **supplier**; and
- **accessPoints** as appropriate.

Add the Continuation Reference to the list of Continuation References in **NRcontinuationList**, and return with **entry unsuitable**.

10) If the last entry found contains a NSSR (**DSE(lastEntryFound)** is of type **nssr**), then generate a Continuation Reference from the NSSR knowledge found in **DSE(lastEntryFound)**. Set:

- **targetObject** to the name of the target object constructed from the resolved components using primary RDNs (alternative distinguished values may be included in the RDNs) concatenated with the remaining unresolved components;
- **operationProgress.nameResolutionPhase** to **proceeding**;
- **operationProgress.nextRDNToBeResolved** to **lastEntryFound+1**;
- **referenceType** to **nonSpecificSubordinate**; and
- **accessPoints** as appropriate.

Add the Continuation Reference to the list of Continuation References in **candidateRefs**. Continue at step 7).

If **DSE(lastEntryFound)** is not of type **nssr**, then continue at step 6).

11) If **chainingArguments.referenceType** is of type **nssr**, then continue at step 13), otherwise at step 12).

12) Return **serviceError** with problem **invalidReference**.

13) If **i + 1** is equal to **nextRDNToBeResolved**, then the request was routed here due to an NSSR and the DSA is unable to proceed with name resolution; in this case, return **serviceError** with problem **unableToProceed**; otherwise continue at step 12).

18.3.3 Target Found sub-procedure

This sub-procedure is entered when the target object name matches with an entry DSEs locally. This sub-procedure checks if the found entry is suitable for processing the request locally (it is shown in Figure 11):

1) Call the Check Suitability procedure.

2) If the entry is suitable (**entry suitable**), then set **nameResolutionPhase** to **completed** and return **entry suitable**.

3) If the entry is not suitable (**entry unsuitable**), then generate a Continuation Reference using the **supplierKnowledge** attribute found in **DSE(lastCP)**. Set:

- **targetObject** to the name of the target object constructed from the resolved components using primary RDNs (alternative distinguished values may be included in the RDNs) concatenated with the remaining unresolved components;
- **operationProgress.nameResolutionPhase** to **proceeding**;
- **operationProgress.nextRDNToBeResolved** to **m**;
- **referenceType** to **supplier**; and
- **accessPoints** as appropriate.

Add the Continuation Reference to the list of Continuation References in **NRcontinuationList**. Return **entry unsuitable**.

NOTE – If the **localScope** service control is set, however, the DSA could, based on local policies, decide to consider this entry as suitable and proceed as in step 2).

4) If a critical extension is not supported (**unsupported critical extension**), then return **serviceError** with problem **unavailableCriticalExtension**.

18.3.4 Check Suitability procedure

This procedure is called to decide whether a found DSE is suitable for performing the requested operation (see Figure 12). It takes into account the **ChainingArguments**, the **ServiceControls**, the arguments as supplied by the user, the operation type and the characteristics of the DSE (shadow, subordinate knowledge, attributes present, etc.).

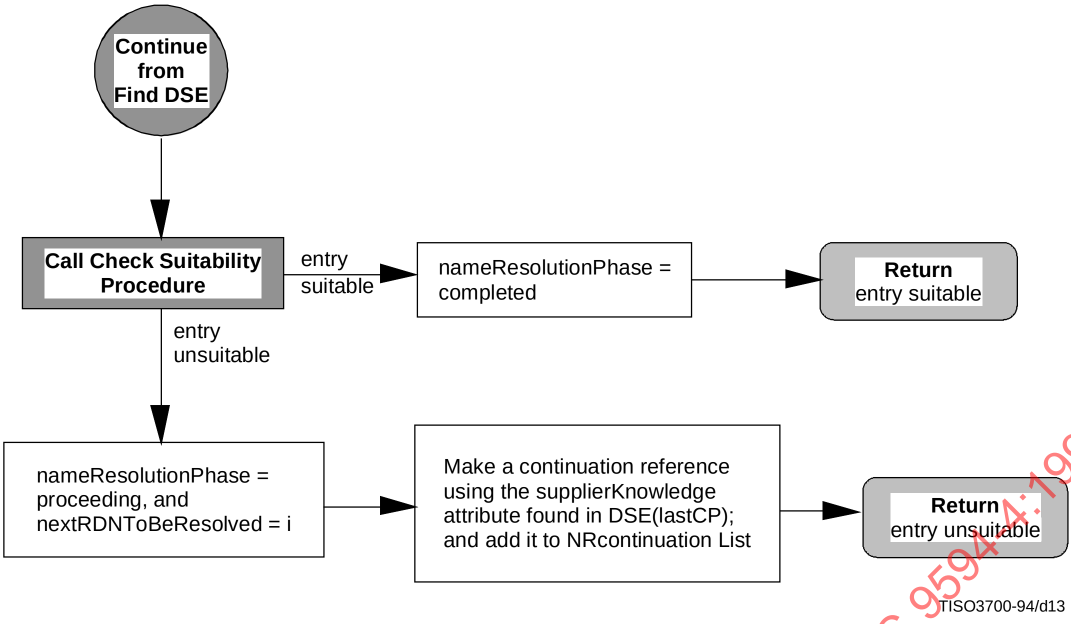


Figure 11 – Target Found sub-procedure

18.3.4.1 Procedure parameters

The input argument to this procedure is:

- a reference to a DSE;
- the operation type for which the suitability of the DSE is to be checked;
- the **ChainingArguments**; and
- the operation argument.

The output is either **entry suitable**, **entry unsuitable**, or **unsupported critical extension**.

1) If the DSE is not of type **shadow**, then check if all **criticalExtensions** are supported. If they are, then return entry suitable, else return unsupported critical extension.

2) The DSE is of type **shadow**. Return entry unsuitable, if any of the following is true:
   - The requested operation type is a modification operation.
   - The service control **dontUseCopy** is set.

   Otherwise, continue with the next step.

3) If the service control **copyShallDo** is set, then check if all **criticalExtensions** are supported. If they are, then return **entry suitable**, else return **unsupported critical extension**.

4) If the service control **copyShallDo** is not set, then check if all **criticalExtensions** are supported. If they are, then got step 5) else return **entry unsuitable**.

5) Distinguish between operation types:

   If List operation, continue at step 6).

   If Read operation, continue at step 7).

   If Search or Compare operation, continue at step 8).

6) If the entry has full subordinate knowledge, the List operation can be performed. In this case, return **entry suitable**, otherwise return **entry unsuitable**.

7) If all the requested attributes are present in the DSE, then return **entry suitable**. If some attributes are missing, then determine by local means whether the shadow copy holds all the attributes held by the master (e.g. by reference to the shadowing agreement). If they are, the entry is suitable (return **entry suitable**). Otherwise, the supplier may hold the requested attributes which are not present at the shadow; in this case, the request has to be chained (return **entry unsuitable**).

8) If the operation is **search** with **searchAliases** set to **TRUE** and the DSE is of type **alias** then if **chainingArguments.excludeShadows** is **FALSE** return **entry suitable**, if it is **TRUE** return **entry unsuitable**.

9) If the DSA supports the matching rule for comparing or searching as requested and the operation is **compare** or **search** operation with **subset** of **baseObject**, then continue at step 7). If the DSA supports the matching rule and the operation is **search** with subset **oneLevel** or **subtree**, then continue at step 10). Otherwise return **entry unsuitable**.

10) If **chainingArguments.excludeShadows** is **TRUE**, then return **entry unsuitable**. Otherwise, check the local understanding of the shadowed information specification against the operation filter and selection. If all necessary entries and attributes are present, then return **entry suitable**. If any entry or attribute is missing, then return **entry unsuitable**.

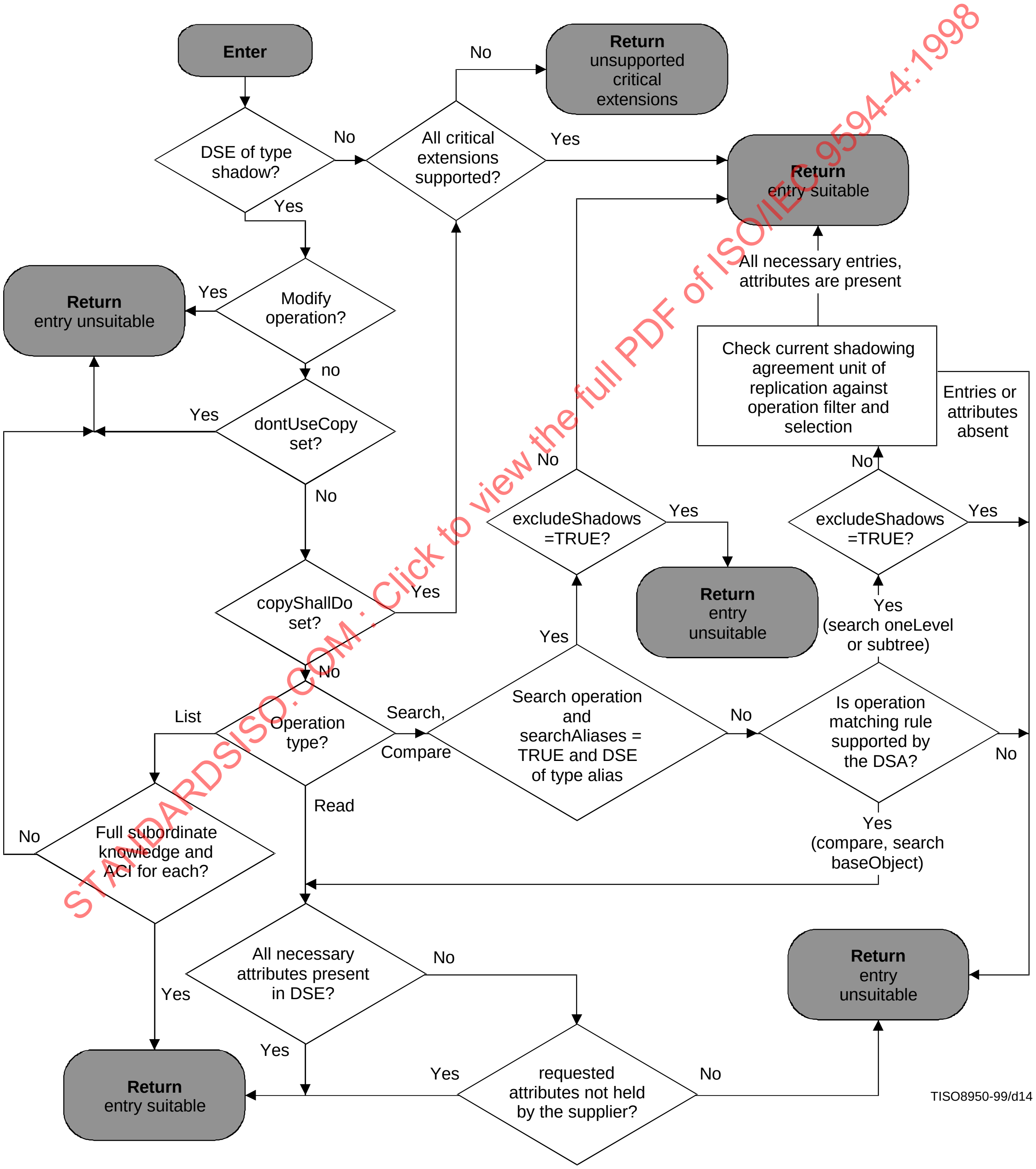


Figure 12 – Check Suitability procedure

19 Operation evaluation

This clause defines the procedure that a DSA shall follow if the target entry of an operation has been found locally (during Name Resolution). According to the type of operation, one of the following procedures are invoked:

– For an **addEntry**, **chainedAddEntry**, **removeEntry**, **chainedRemoveEntry**, **modifyEntry**, **chainedModifyEntry**, **modifyDN** or **chainedModifyDN** operation the procedures in 19.1 shall be followed.

– For a **read**, **chainedRead**, **compare** or **chainedCompare** operation the procedures in 19.2 shall be followed.

– For a **search**, **chainedSearch**, **list** or **chainedList** operation the procedures in 19.3 shall be followed.

19.1 Modification procedure

According to the type of modification operation, the corresponding procedures defined in 19.1.1 through 19.1.4 shall be followed.

19.1.1 Add Entry Operation

1) The DSA shall check that the initiator has sufficient access rights, e.g. as defined, in 11.1.5 of ITU-T Rec. X.511 | ISO/IEC 9594-3. If not, an appropriate error is returned.

2) The DSA shall assure that an entry with the name of the entry to be added does not already exist. Otherwise, it shall return an **updateError** with problem **entryAlreadyExists**. If the superior DSE is of additional type **nssr**, the DSA shall follow the procedure defined in 19.1.5 (Modify Operations and NSSRs) to ensure that the name of the new entry is unambiguous. If the name of the entry to be added includes multiple distinguished values differentiated by context for some attribute in the final RDN, the DSA shall assure that none of the possible alternative RDNs that may be constructed yields (regardless of context) a name for an entry that already exists.

3) If **targetSystem** is present, and the **AccessPoint** is not that of the current DSA, go to step 4). If **targetSystem** is not present, or is present and the **AccessPoint** is that of the current DSA, go to step 5).

4) If the entry is a subentry, the DSA shall return **updateError** with problem **affectsMultipleDSAs**. If the entry is not a subentry, the DSA has a local choice as to whether or not it wishes to establish a HOB with the specified DSA. If it does not, the DSA shall return **serviceError** with problem **unwillingToPerform**, otherwise the DSA shall establish a hierarchical operational binding (HOB) with the specified subordinate DSA. If the DOP is supported, the procedure in 24.3.1.1 shall be followed. Otherwise, local means are used to establish the HOB. If the subordinate DSA is unwilling to establish the operational binding, a **serviceError** with problem **unwillingToPerform** is returned for the **addEntry** operation. If the HOB is successfully established, continue at step 7).

   NOTE 1 – This step of the procedure does not apply to the creation of autonomous administrative areas in a subordinate DSA.

5) The DSA shall ensure that the new entry conforms to the sub-schema, or that the new subentry or DSE of other types conform to the system schema (e.g. that the immediate superior DSE of a subentry is of type **admPoint**). If not, it shall return an appropriate **updateError** or **attributeError**, else it shall add the new DSE. If entry, continue at step 7). If subentry, continue at step 6). Otherwise, appropriate knowledge management procedures for the other types of DSE are executed. See Section 6.

6) The DSA shall forward, at an appropriate time, a modify operational binding to all relevant subordinate DSAs with which it has hierarchical or non-specific hierarchical operational bindings. The relevant bindings are those which are associated with naming contexts that are subordinate to the superior DSE. Naming contexts whose context prefixes correspond to autonomous administrative points are not relevant. If the DOP is supported, the procedures in 24.3.2.1 and 25.3.2 shall be followed. If the DOP is not supported, local means shall be used to modify the RHOBs.

   NOTE 2 – An appropriate time is specified by the DSA administrator, and might range from immediately after (or even before) the operation result is returned to a periodic strategy (e.g. at an appointed hour). The time may vary depending upon the reason for the modification, e.g. updates to ACI taking immediate effect and changes to schema being done periodically.

7) If the added entry or subentry is within the **UnitOfReplication** of one or more shadowing agreements, then the shadow consumers shall be updated using the procedures of the Directory information shadow service specified in ITU-T Rec. X.525 | ISO/IEC 9594-9.

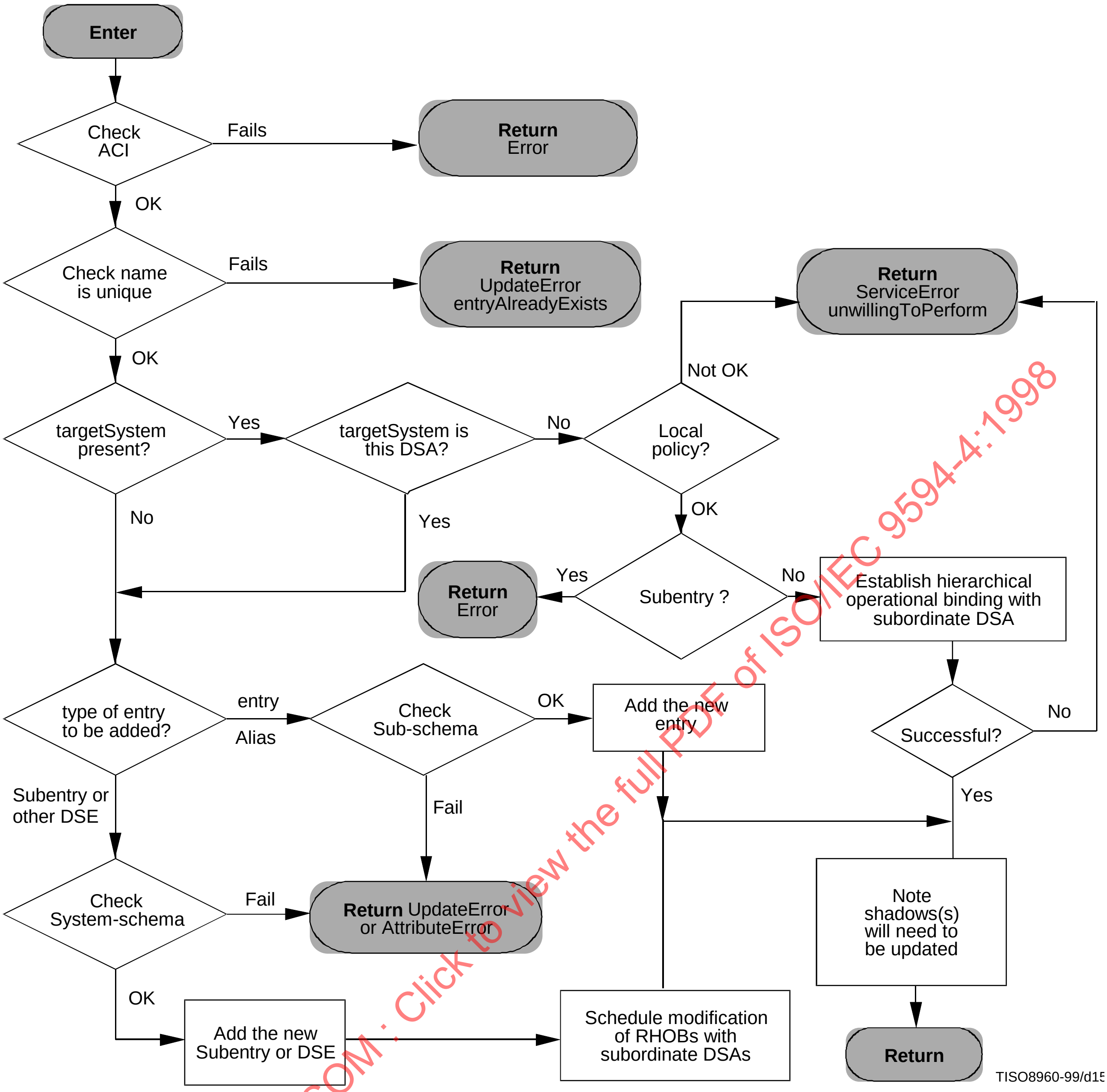


Figure 13 – Add Entry

19.1.2 Remove Entry Operation

1) The DSA shall check that the initiator has sufficient access rights, e.g. as defined, in 11.2.5 of ITU-T Rec. X.511 | ISO/IEC 9594-3. If not, an appropriate error is returned.

2) The DSA shall ensure that the entry to be removed is a leaf entry. Otherwise, the DSA shall return an **updateError** with problem **notAllowedOnNonLeaf**.

3) The DSE type of the entry to be removed is checked. If **subentry**, continue at step 5). If **cp**, continue at step 6). If **entry** or **alias**, continue at step 4). Otherwise, appropriate knowledge management procedures for the other types of DSE are executed. See Section 6.

4) Remove the entry or alias entry and continue at step 7).

5) Remove the subentry. At an appropriate time, modify the operational bindings of all relevant subordinate DSAs with which the current DSA has hierarchical or non-specific hierarchical operational bindings. The relevant bindings are those which are associated with naming contexts subordinate to the superior DSE.

Naming contexts whose context prefixes correspond to autonomous administrative points are not relevant. If the DOP is supported, the procedures in 24.3.2.1 and 25.3.2 shall be followed. Otherwise, local means shall be used. Continue at step 7).

6) Remove the naming context. If the DSA has a hierarchical operational binding for this naming context, it shall terminate the hierarchical operational binding with its immediately superior DSA. If the DSA has a non-specific hierarchical operational binding for this naming context, and this is the last naming context of the non-specific hierarchical operational binding, then it shall terminate the non-specific hierarchical operational binding with its immediately superior DSA. If the DOP is supported, the procedures in 24.3.3.2 and 25.3.3.2 shall be followed. Otherwise, local means are used to terminate the RHOB.

7) If the removed naming context, entry, alias entry or subentry was within the **UnitOfReplication** of one or more shadowing agreements, then the shadow consumers shall be updated using the procedures of the Directory information shadow service specified in ITU-T Rec. X.525 | ISO/IEC 9594-9.

If the removed subordinate or non-specific subordinate reference in the immediately superior DSA (whose RHOB was terminated), was within the **UnitOfReplication** of one or more shadowing agreements, then the shadow consumers shall be updated using the procedures of the Directory information shadow service specified in ITU-T Rec. X.525 | ISO/IEC 9594-9.

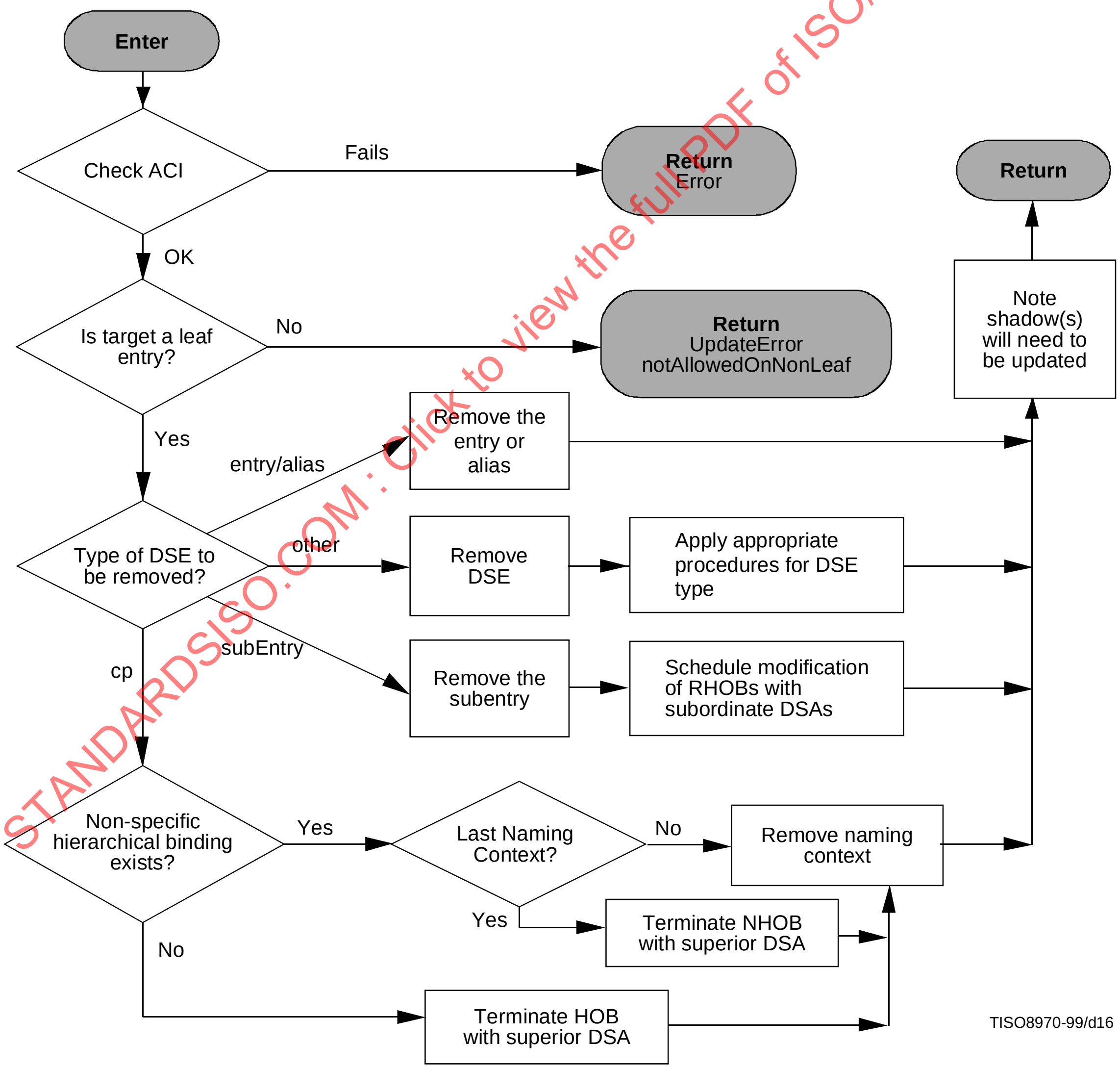


Figure 14 – Remove Entry procedure

19.1.3 Modify Entry Operation

1) The DSA shall check that the initiator has access rights, e.g. as defined, in 11.3.5 of ITU-T Rec. X.511 | ISO/IEC 9594-3. If not, an appropriate error is returned.

2) The modifications to the entry or alias shall conform to the sub-schema. The modification to a DSE of other types, including subentry, shall conform to the system schema. Otherwise, the DSA shall return an appropriate **updateError** or **attributeError**. After performing the modifications, if the target DSE is of type **subentry**, continue at step 3); if the target DSE is of type **entry** or **alias**, continue at step 4); otherwise, appropriate knowledge management procedures for the other types of DSE are executed. See Section 6.

3) The DSA shall, at an appropriate time, modify the operational bindings with all relevant subordinate DSAs with which it has hierarchical or non-specific hierarchical operational bindings. The relevant bindings are those which are associated with naming contexts that are subordinate to the administrative point that the modified subentry is located below. Naming contexts whose context prefixes correspond to autonomous administrative points are not relevant. If the DOP is supported, the procedure in 24.3.2.1 and 25.3.2 shall be followed. Otherwise, local means are used.

4) If the modified entry, alias entry or subentry was within the **UnitOfReplication** of one or more shadowing agreements, then the shadow consumers shall be updated using the procedures of the Directory information shadow service specified in ITU-T Rec. X.525 | ISO/IEC 9594-9.

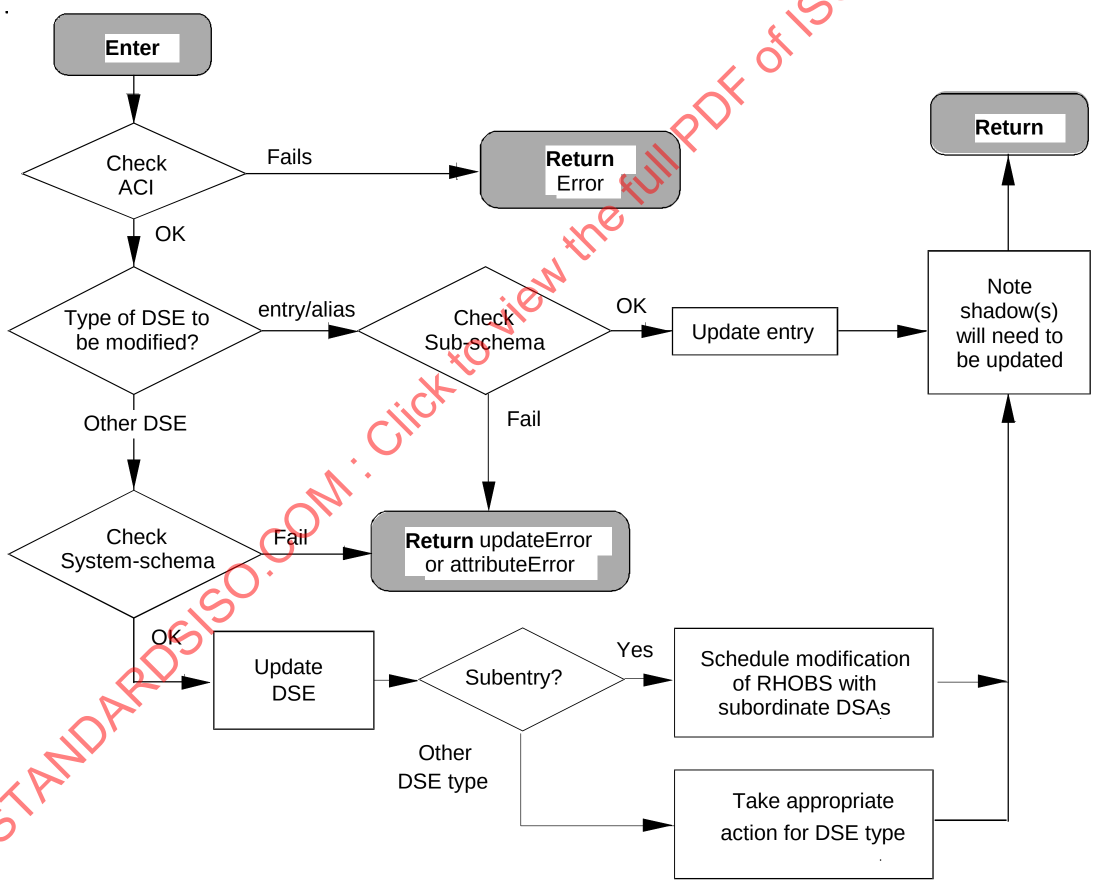


Figure 15 – Modify Entry

19.1.4 Modify DN operation

1) The DSA shall check that the initiator has sufficient access rights, e.g. as defined in 11.4.5 of ITU-T Rec. X.511 | ISO/IEC 9594-3. If not, an appropriate error is returned.

2) If the operation is either to move an entry or to both move an entry and change its Relative Distinguished Name, go to step 3). If the operation is to only change the Relative Distinguished Name of an entry, go to step 4).

3) The operation shall be performed according to the definition in 11.4.1 of ITU-T Rec. X.511 | ISO/IEC 9594-3. If either the old superior, the new superior, the entry or any of its subordinates are not in this DSA, or if the new superior has NSSRs, then the operation shall be rejected with **updateError** with problem **affectsMultipleDSAs**. The DSA shall ensure that no other entry with the new name already exists. Otherwise, it shall return an **updateError** with problem **entryAlreadyExists**. The DSA shall ensure that the new name of the entry conforms to the subschema. Otherwise, it shall return an appropriate **attributeError** or **updateError**. If none of these problems arise then move the entry (changing the RDN if required) and go to step 9).

4) The following text is applicable to changing the relative distinguished name of an entry, which may or may not be a leaf entry, and which may or may not have one or more subordinates in one or more DSAs. The DSE type of the entry to be renamed is checked. If **subentry**, continue at step 7). If **cp**, continue at step 6). If **entry** or **alias**, continue at step 5).

5) The DSA shall ensure that no other entry with the new name already exists. Otherwise, it shall return an **updateError** with problem **entryAlreadyExists**. If the superior DSE of the entry to be renamed is of additional type **nssr**, the DSA shall follow the procedure defined in 19.1.5 (Modify Operations and NSSRs) to ensure that the new name of the entry is unambiguous. If the new name includes multiple distinguished values differentiated by context for some attribute in an RDN, the DSA shall assure that none of the possible RDNs that may be constructed yields (regardless of context) a name for an entry that already exists. The DSA shall ensure that the new name of the entry conforms to the subschema. Otherwise, it shall return an appropriate **attributeError** or **updateError**. Rename the entry or alias entry. If the entry is a non-leaf entry and has subordinates in other DSAs, continue at step 8), otherwise continue at step 9).

6) The DSA shall ensure that the new name of the naming context conforms to the subschema; otherwise, it shall return an appropriate **attributeError** or **updateError**.

   If the DSA has a HOB with the superior DSA, then the subordinate DSA shall attempt to modify the HOB before responding to the Modify DN operation. The superior DSA shall ensure that no other entry with the new name already exists, before accepting the modification. If the DOP is supported, the procedure in 24.3.2.2 shall be followed. If the DOP is not supported, it is a local matter how the HOB is modified and the new name is checked for uniqueness. If the HOB is successfully modified, and the naming context has subordinate naming contexts in other DSAs, go to step 8); otherwise, go to step 9). If the HOB cannot be modified return **updateError** with problem **affectsMultipleDSAs**.

   If the DSA has a NHOB for this naming context with the superior DSA, then how duplicate entries are detected is outside the scope of this Directory Specification. Rename the entry. If the naming context has subordinate naming contexts in other DSAs, go to step 8); otherwise, go to step 9).

7) The DSA shall ensure that the new name of the subentry conforms to the system schema. Otherwise, it shall return an appropriate **attributeError** or **updateError**. The DSA shall ensure that no other subentry with the new name already exists. Otherwise, it shall return an **updateError** with problem **entryAlreadyExists**.

8) The DSA shall, at an appropriate time, modify the operational bindings of all relevant subordinate DSAs with which it has hierarchical or non-specific hierarchical operational bindings. The relevant bindings are those which are associated with all naming contexts that are subordinate to the entry being renamed, or relevant naming contexts that are subordinate to the administrative point whose subentry was renamed. Naming contexts whose context prefixes correspond to autonomous administrative points are not relevant. If the DOP is supported, the procedures in 24.3.2.1 and 25.3.2 shall be followed. Otherwise, local means shall be used to update the RHOBs.

9) If the renamed naming context, entry or any of its subordinates, alias entry or subentry is within the **UnitOfReplication** of one or more shadowing agreements held by the DSA, then the shadow consumers shall be updated using the procedures of the Directory information shadow service specified in ITU-T Rec. X.525 | ISO 9594-9.

   If the renamed subordinate reference in the immediately superior DSA [whose HOB was modified in step 6) above] is within the **UnitOfReplication** of one or more of its shadowing agreements, then the shadow consumers shall be updated using the procedures of the Directory information shadow service specified in ITU-T Rec. X.525 | ISO/IEC 9594-9.

   If components of a RHOB with a subordinate DSA [as modified in step 8) above] are within the **UnitOfReplication** of one or more shadowing agreements held by the subordinate DSA, then the shadow consumers shall be updated using the procedures of the Directory information shadow service specified in ITU-T Rec. X.525 | ISO/IEC 9594-9.

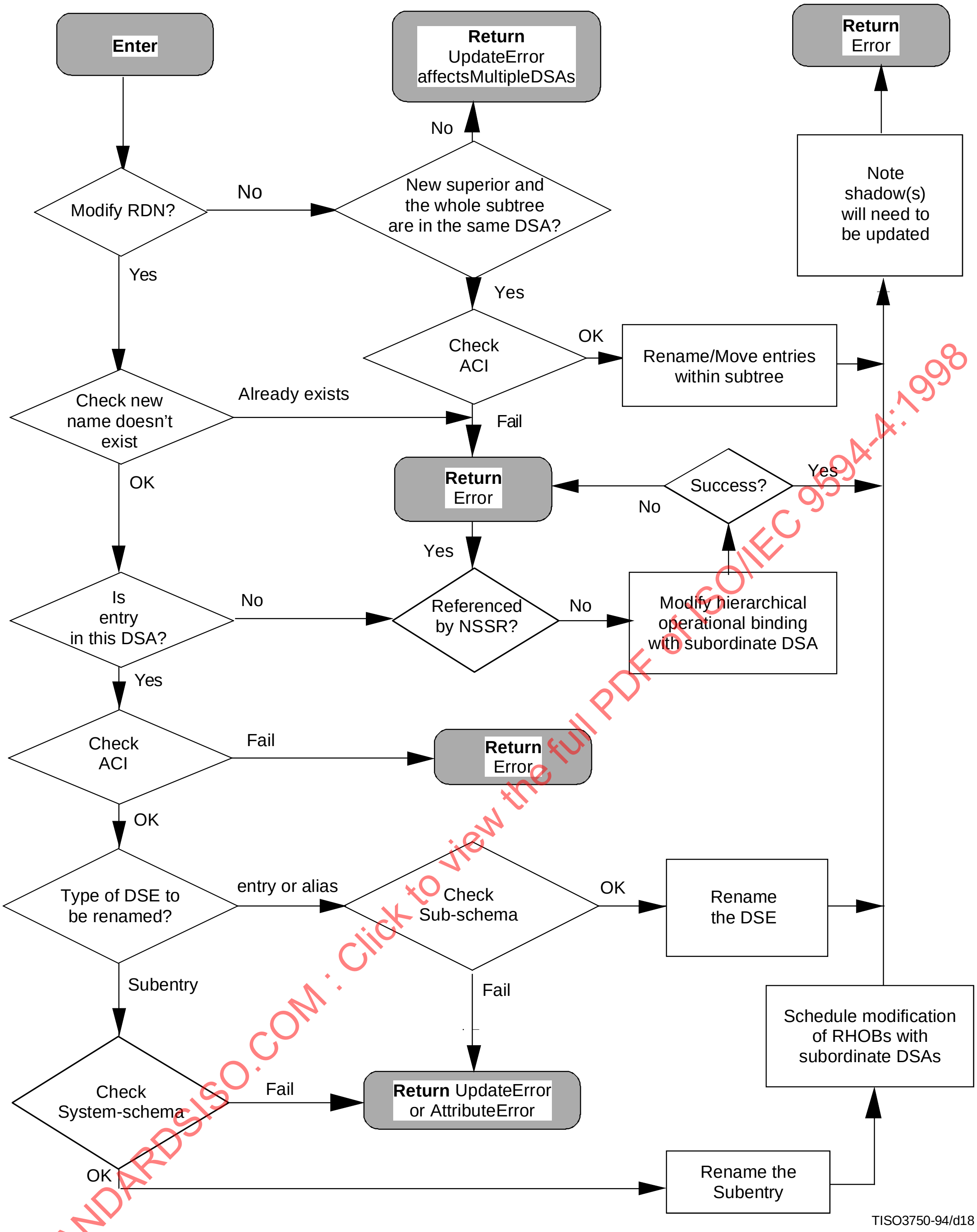


Figure 16 – Modify DN procedure

19.1.5 Modify operations and Non-Specific Subordinate References

If a DSA has NSSRs and does not know the complete set of names of the subordinates of an entry, to which either

a) an **addEntry** operation has been directed; or

b) a **modifyDN** operation has been directed;

then the DSA may perform the following set of procedures prior to performing the operation.

1) If the **chainingProhibited** service control option is set on the **addEntry** or **modifyDN** operation, return **updateError** with problem **affectsMultipleDSAs**.

2) If the DSA is unwilling or unable to multi-chain outgoing requests, return **serviceError** with problem **unwillingToPerform** or **unavailable**, respectively.

3) The DSA shall multi-chain a **chainedReadEntry** operation to each master DSA in the set of **accessPointInformation** of the NSSR. (The DSA shall only use the master DSA from each **MasterAndShadowAccessPoints** due to transient inconsistency caused by shadowing.) The parameters of the **ReadArgument** shall be set as follows:

   object to either the name of the entry to be added (in the case of **addEntry**), or to the proposed name of an existing entry (in the case of **modifyDN**).

   selection the object class attribute.

   The parameters of **CommonArguments** shall be set as follows:

   – set the **dontDereferenceAliases** service control option;

   – set **OperationProgress.nameResolutionPhase** to **completed**.

   The parameters of **ChainingArguments** shall be set as follows:

   – set **originator** to the name of the originator;

   – **targetObject** is omitted;

   – set **OperationProgress.nameResolutionPhase** to **proceeding** and **nextRDNToBeResolved** to (number of RDNs in the object name) – 1;

   – set **traceInformation** to an empty sequence;

   – set **referenceType** to **nonSpecificSubordinate**;

   – **timeLimit**, as appropriate according to the incoming request.

   Other parameters, e.g. **SecurityParameters**, may be set as appropriate, e.g. by local policy.

4) The DSA waits for the complete set of responses. If any of the response is a **ReadResult**, then an error shall be returned as in 6) below.

5) If all responses are **serviceError** with problem **unableToProceed**, operation evaluation may proceed.

6) If a **ReadResult** is returned, an **updateError** with problem **entryAlreadyExists** shall be returned for the original operation;

7) If any other error is returned to the **readEntry** request, a **serviceError** with problem **unwillingToPerform** shall be returned.

The DSA receiving the **chainedRead** request shall give a response according to the presence or not of the entry, and its access control policy.

19.2 Single entry interrogation procedure

The operations **read**, **chainedRead**, **compare**, and **chainedCompare** fall into the group of single entry interrogation procedures. These procedures contain only the following three steps:

1) Check access control, as described in clause 9 of ITU-T Rec. X.511 | ISO/IEC 9594-3. If the operation is disallowed, return the appropriate security error.

2) Perform the operation on the found DSE as described in clause 9 of ITU-T Rec. X.511 | ISO/IEC 9594-3.

3) Prepare the reply, and return.

19.3 Multiple entry interrogation procedure

According to the type of interrogation operation (**list** or **search**), the corresponding procedures defined in 19.3.1 and 19.3.2 shall be followed.

19.3.1 List procedures

This subclause specifies the evaluation procedure specific to **list** and **chainedList** operations.

The **List (I)** Procedure shall be followed when the List request's **operationProgress.nameResolutionPhase** component is set to **notStarted** or **proceeding** and when the DSA, after performing Name Resolution, finds that it holds the base object. The **List (II)** Procedure shall be followed when the List request's **nameResolutionPhase** component is set to **completed**.

19.3.1.1 Procedure parameters

19.3.1.1.1 Arguments

The arguments that are used by this procedure are:

- the **ListArgument**;
- the target DSE **e**;
- **operationProgress** of the **chainingArgument**.

19.3.1.1.2 Results

If this procedure is successfully executed, it returns:

- a set of subordinates of **e** in **listInfo.subordinates**;
- **limitProblem** indicated in **partialOutcomeQualifier**;
- a set of continuation references in **SRcontinuationList**.

19.3.1.1.3 Errors

The procedure can result in one of the following errors returned to the requesting DUA/DSA:

- an **accessControlError** or **nameProblem**;
- any error defined for the **find DSE** procedure, when an alias has been dereferenced.

19.3.1.2 Procedure definition

The sub-procedures as defined in 19.3.2.2.1 and 19.3.2.2.2 shall be invoked according to the following rules.

19.3.1.2.1 List (I) procedure

The **List (I)** procedure consists of the following steps as depicted in Figure 17:

1) If the service control **subentry** is set, then go to Step 5); otherwise, go to Step 2).

2) If DSE **e** is of type **nssr**, then add a Continuation Reference to **SRcontinuationList** with the following components:
   - **targetObject** to the primary distinguished name of the DSE **e** (alternative distinguished values may be included in the RDNs);
   - **aliasedRDNs** absent;
   - **operationProgress** with **nameResolutionPhase** set to **completed** and **nextRDNtoBeResolved** absent;
   - **rdnsResolved** absent;
   - **referenceType** set to **nonSpecificSubordinate**;
   - **accessPoints** set to a set of **accessPointInformation** each derived from a value of the **nonSpecificKnowledge** attribute of DSE **e**.

3) For each DSE **e'** immediately subordinate to DSE **e** execute the following steps:
   a) Check the ACI in **e'** if available. If the ACI disallows listing the RDN of **e'**, then skip this DSE. If the ACI is not available (for example in the case of subordinate references and glue), then it is a local policy whether to proceed.
   b) Check all the DSE types of **e'**.
      i) If **e'** is of type **subr**, then there are two cases. In the first case, the subordinate entry's ACI and object class is available locally, in which case, based on local policy and the ACI's permission, add the RDN of **e'** to **listInfo.subordinates** with **aliasEntry** set to **TRUE** if **e'** is of type **sa**, and **fromEntry** set **FALSE**. The other case is when the ACI of the entry is not available in **e'**, in which case add a Continuation Reference to **SRcontinuationList** with the following components:
         - **targetObject** to the primary distinguished name of the DSE **e** (alternative distinguished values may be included in the RDNs);
         - **aliasedRDNs** absent;
         - **operationProgress** with **nameResolutionPhase** set to **completed** and **nextRDNtoBeResolved** absent;

- **rdnsResolved** absent;
- **referenceType** set to **subordinate**;
- **accessPoints** set to the value contained in the **specificKnowledge** attribute of DSE **e'**.

ii) If the DSE **e'** is of type **entry** or **glue**, then add the RDN of **e'** to **listInfo.subordinates** with **aliasEntry** set to **FALSE** and **fromEntry** set according to whether **e'** is a copy.

NOTE – In the case that **e'** is **glue**, it must have one or more subordinates which implies it cannot be an alias in the master DSA. Also, any ACI relevant to the List operation is stored in this DSE, supplied via the shadowing protocol.

iii) If the DSE **e'** is of type **alias**, then add the RDN of **e'** to **listInfo.subordinates** with **aliasEntry** set to **TRUE**, and **fromEntry** set according to whether **e'** is a copy.

c) Check if time, size or administrative is exceeded. If so, set **limitProblem** accordingly in **partialOutcomeQualifier** and return.

d) Continue from step 3) a) until all subordinate DSEs have been processed.

4) If all subordinates DSEs have been processed, return to the **Operation Dispatcher**.

5) For each subentry **e'** immediately subordinate to DSE **e** execute the following steps:

a) Check the ACI in **e'**. If the ACI disallows listing the RDN of **e'**, then skip this DSE. Otherwise, add the RDN of **e'** to **listInfo.subordinates** with **aliasEntry** set to **FALSE** and **fromEntry** set according to whether **e'** is a copy.

b) Check if time, size or administrative limit is exceeded. If so, set **limitProblem** accordingly in **partialOutcomeQualifier** and return.

6) Return to the **Operation Dispatcher**.

19.3.1.2.2 List (II) procedure

The **List (II)** procedure consists of the following steps as depicted in Figure 18:

1) For each DSEs **e'** immediately subordinate to DSE **e,** execute steps 1), a) to 1), d):

a) Check ACI in **e'**. If the operation is disallowed by the ACI, continue with the next immediate subordinate of **e**.

b) Add the RDN of DSE **e'** to **listInfo.subordinates**, with the **aliasEntry** component of **listInfo.subordinates** to according to whether **e'** is an alias, and the **fromEntry** component set depending on whether **e'** is a copy or not. Ignore those DSEs of type **shadow**, if **excludeShadows** is **TRUE**.

c) Check if time, size or administrative limit is exceeded. If so, set the **limitProblem** of **partialOutcomeQualifier** accordingly and return.

d) continue from step 1) a) until all subordinate DSEs have been processed.

2) If all subordinate DSEs have been processed, check if this subrequest came from DAP or DSP. In case this subrequest is submitted via DAP, and the **ListResult** is empty, then return a **serviceError** with problem **invalidReference** to the Operation Dispatcher. Otherwise, the **ListResult** is returned.

NOTE – **invalidReference** is used as a security precaution in case the user does not have access to the superior entry. If the superior's entry ACI is available (provided by the RMOB), then a null result may be returned if allowed.

19.3.2 Search procedures

This subclause specifies the evaluation procedure specific to **search** and **chainedSearch** operations.

Search (I) procedure shall be followed when the **search** request's **operationProgress.nameResolutionPhase** component is set to **notStarted** or **proceeding** and when the DSA, after performing Name Resolution, finds that it holds the base object. The **Search (II)** procedure shall be followed when the **search** request's **nameResolutionPhase** component is set to **completed**.

NOTE – When **nameResolutionPhase** is **completed**, the target object is expected to be the immediate superior of a context prefix.

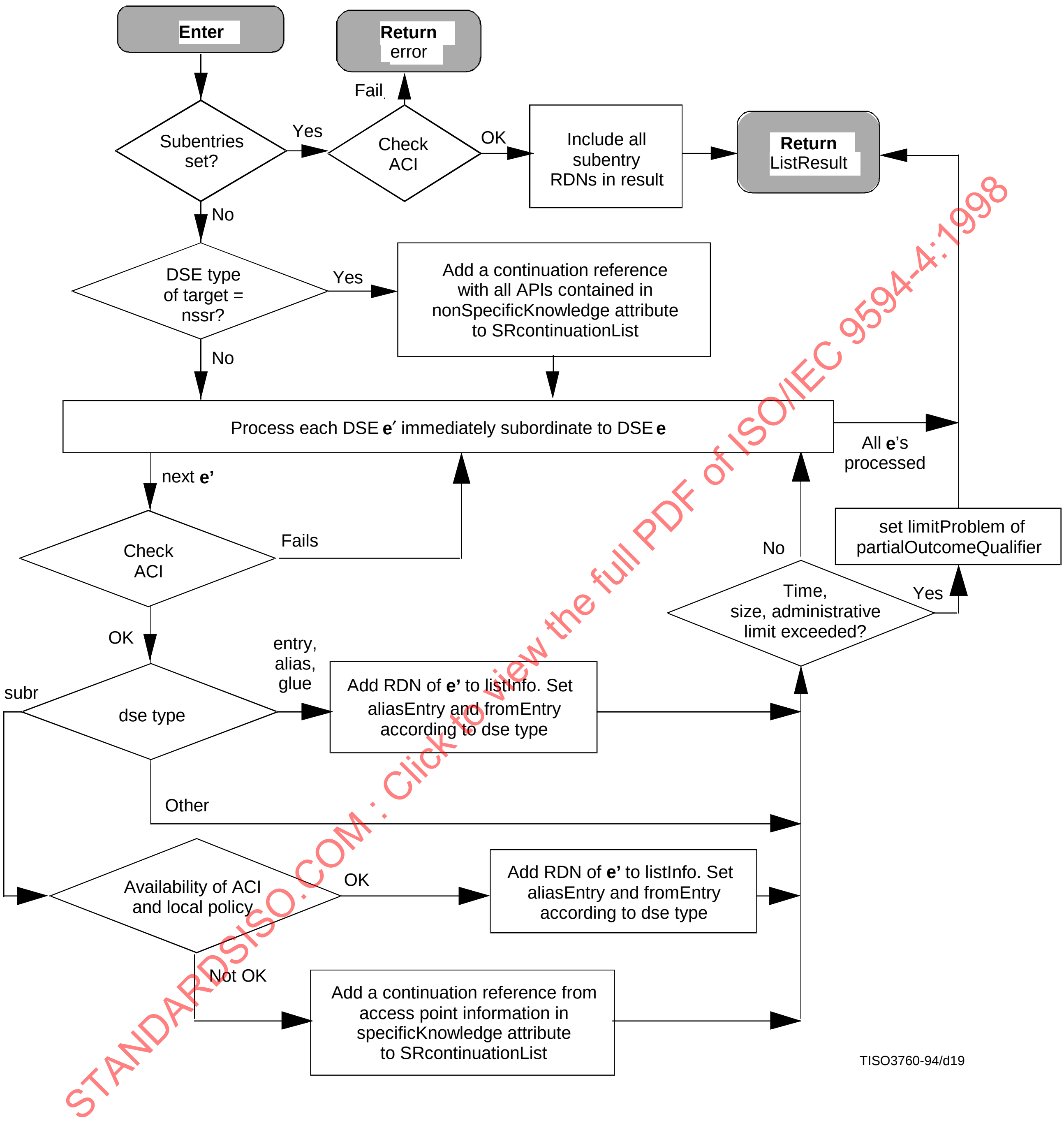


Figure 17 – List (I) procedure

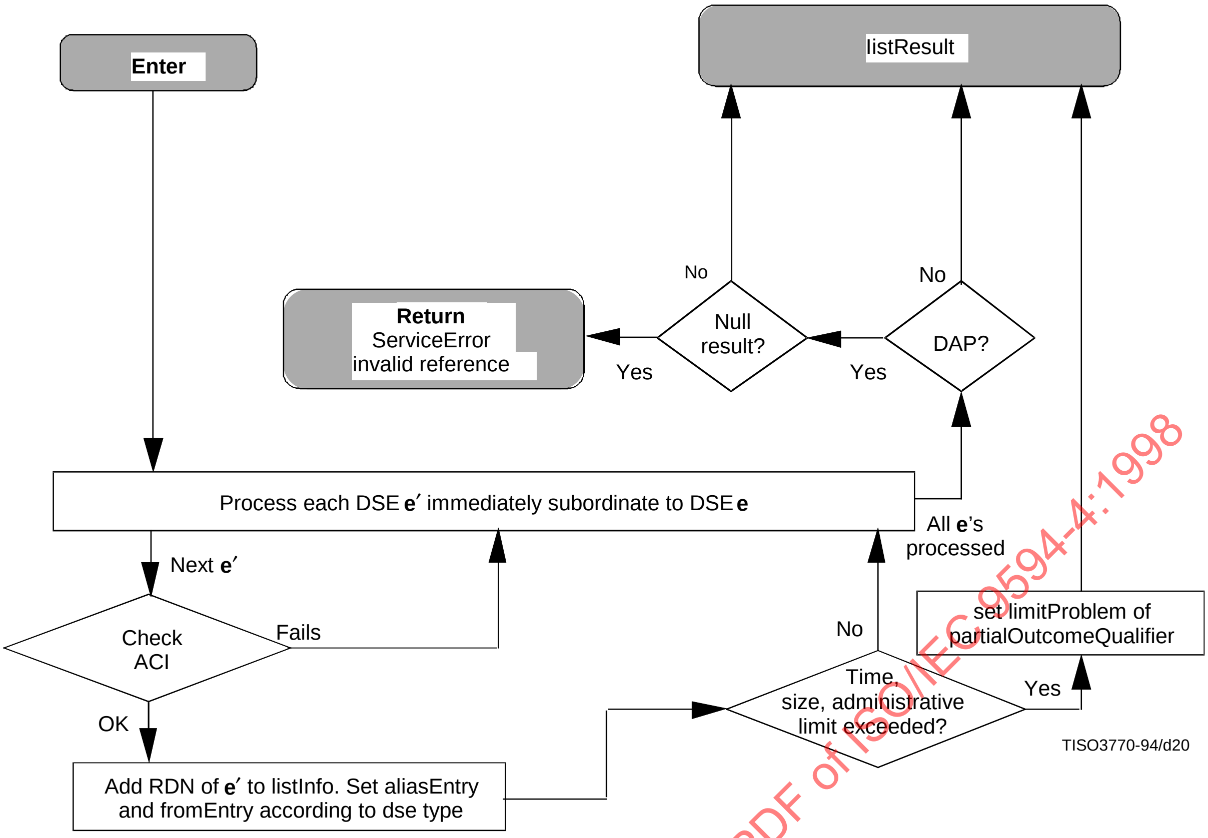


Figure 18 – List (II) procedure

19.3.2.1 Procedure parameters

19.3.2.1.1 Arguments

The arguments that are used by this procedure are:

– the **SearchArgument**;

– the target DSE **e**;

– **operationProgress** of the **ChainingArguments**;

– **exclusions** of the **ChainingArguments** (a list of RDNs to exclude from search).

19.3.2.1.2 Results

If this procedure is successfully executed, it returns:

– a set of matched Entries in **searchResult.entryInformation**;

– **alreadySearched** in **ChainingResults**;

– a set of continuation references in **SRcontinuationList**.

19.3.2.2 Procedure definition

19.3.2.2.1 Search (I) procedure

This is a recursive procedure that applies to a **search** request that starts at a given target entry **e**. It searches the target entry **e** and then processes the DSEs immediately subordinate to **e**. The procedure is invoked by itself recursively in the case that a whole subtree is to be searched. The procedure consists of the following steps as shown in Figure 19:

1) If the type of DSE **e** is of type **cp** (a DSE at a context prefix), check if any element of the **exclusions** argument is a prefix of the DN of **e**.

   a) If so, return.

b) Else, call Check Suitability.

i) If **e** is unsuitable, make a Continuation Reference as follows and add it to **SRcontinuationList**:

– **targetObject** to the primary distinguished name of the immediate superior of DSE **e** (alternative distinguished values may be included in the RDNs);

– **aliasedRDNs** absent;

– **operationProgress** with **nameResolutionPhase** set to **completed** and **nextRDNtoBeResolved** absent;

– **rdnsResolved** absent;

– **referenceType** set to **supplier**;

– **accessPoints** set to **AccessPointInformation** derived from the value(s) found in the **supplierKnowledge** attribute in **e**.

Then return.

NOTE 1– This is the only place when a **search** subrequest (**nameResolutionPhase** is **completed**) is chained to a shadow's supplier. In other words, the target object for such a chained subrequest is always a context prefix.

ii) Else, add the Distinguished Name of **e** to **alreadySearched** in **ChainingResults**.

NOTE 2 – **alreadySearched** only contains context prefixes.

2) If **e** is of type **alias** and **searchAliases** in **SearchArgument** is **TRUE** then call **Search Alias** procedure and then return.

3) If **subset** is **oneLevel**, then proceed to step 6).

NOTE 3 – The **e** cannot be subordinate incomplete at this point since the Check Suitability at the context prefix should have ascertained that this cannot happen.

4) If **subset** is **baseObject**, or if **entryOnly** is **TRUE** then continue with this step; otherwise, go to step 5).

If one of the following is true:

a) **e** is of type **subentry** and the service control **subentry** is set; or

b) **e** is not of type **subentry** and the service control **subentry** is not set,

then do the following steps:

i) Check ACI. If the operation is disallowed, return.

ii) Apply the filter argument specified in the **SearchArgument.filter** to the DSE **e**. Ensure that access to all attributes used in the filter is permitted as defined in ITU-T Rec. X.501 | ISO/IEC 9594-2. If the filter matches, add the attributes selected by the **SearchArgument.selection** to the list of matched entries in **searchResult.entryInformation**. Only add attributes that are not greater than the **attributeSizeLimit**.

Return.

5) If **subset** is **subtree** (and **entryOnly** is not **TRUE**), and in addition one of the following is true:

a) **e** is of type **subentry** and the service control **subentry** is set; or

b) **e** is not of type **subentry** and the service control **subentry** is not set,

then do the following steps:

i) Check ACI. If the operation is disallowed, go to step 6).

ii) Apply the filter argument specified in the **SearchArgument.filter** to the DSE **e**. Ensure that access to all attributes used in the filter is permitted as defined in ITU-T Rec. X.501 | ISO/IEC 9594-2. If the filter matches, add the attributes selected by the **SearchArgument.selection** to the list of matched entries in **searchResult.entryInformation**.

iii) Proceed to step 6).

6) If **e** is of type **nssr**, then add a Continuation Reference to **SRcontinuationList** with the following components:

   – **targetObject** to the primary distinguished name of the DSE **e** (alternative distinguished values may be included in the RDNs);

   – **aliasedRDNs** absent;

   – **operationProgress** with **nameResolutionPhase** set to **completed** and **nextRDNtoBeResolved** absent;

   – **rdnsResolved** absent;

   – **referenceType** set to **nssr**;

   – **accessPoints** set to **AccessPointInformation** derived from the value(s) found in the **nonSpecificKnowledge** attribute.

7) Process all DSEs **e'** that are located immediately subordinate to the target DSE **e** until all subordinate DSEs have been processed. During this loop, if the list of matched entries in **searchResult.entryInformation** exceeds the size limit, or time or administrative limit is exceeded then set **limitProblem** accordingly in **partialOutcomeQualifier** and return.

   NOTE 4 – The check for size limit is also implicitly applied every time **searchResult** is updated.

   a) If the DSE **e'** is of type **subr**, and not of type **cp**, then add a Continuation Reference to **SRcontinuationList** with the following components:

      – **targetObject** to the primary distinguished name of the DSE **e** (alternative distinguished values may be included in the RDNs);

      – **aliasedRDNs** absent;

      – **operationProgress** with **nameResolutionPhase** set to **completed** and **nextRDNtoBeResolved** absent;

      – **rdnsResolved** absent;

      – **referenceType** set to **subr**;

      – **accessPoints** set to the access point information contained in the **specificKnowledge** attribute of DSE **e'**.

      NOTE 5 – If **e'** is of both type **cp** and **subr**, a search subrequest can be generated potentially from either the subordinate reference or the supplier knowledge, but not both. This procedure uses the latter (supplier references found in cp).

   b) For all cases:

      i) If **subset** is **oneLevel**, set **entryOnly** to **TRUE**.

      ii) Recursively execute **Search (I)** procedure for target DSE **e'**.

8) If all subordinates have been processed, return to the operation dispatcher for further processing.

19.3.2.2.2 Search (II) procedure

This procedure applies if a search request is processed that originated from a request decomposition at the DSA from which the request was received. The procedure processes the DSEs below the target DSE **e** and calls the **Search (I)** procedure for each object entry:

1) Process all DSEs **e'** that are located immediately subordinate to the target DSE **e** until all subordinate DSEs have been processed. When all subordinates have been processed, return to the operation dispatcher for further processing.

2) If the DSE is not of type **cp** then ignore it. Return to step 1).

3) Call **Check Suitability**. If suitable go to step 4); otherwise, ignore it and return to step 1).

4) Execute the **Search Procedure (I)** for the DSE **e'** as described in 19.3.2.2. If the DSE is of type **alias** and the value of the **subset** parameter is set to **oneLevel**, set **ChainingArguments.entryOnly** to **TRUE** when calling **Search (I)** procedure. Return to step 1).

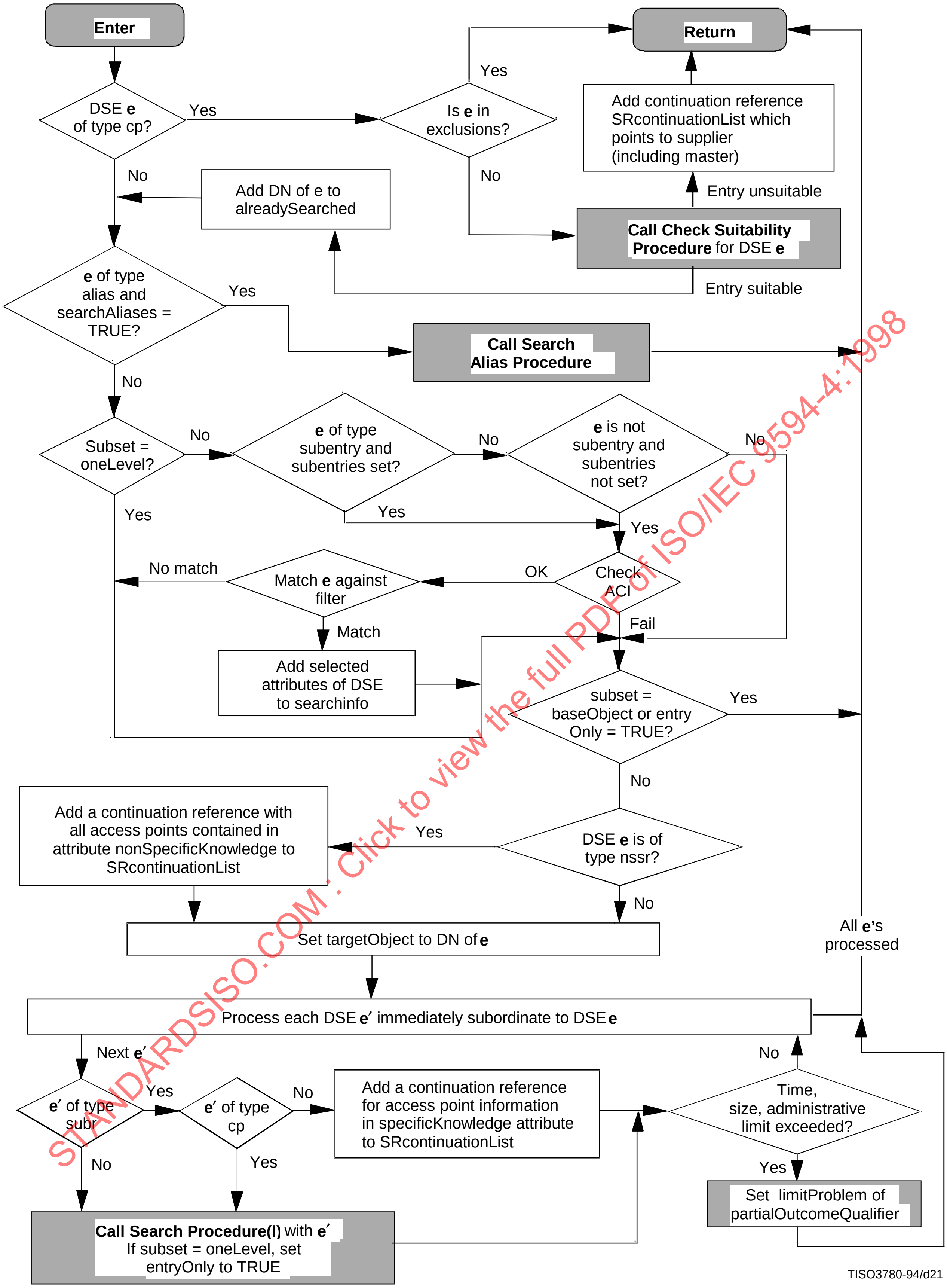


Figure 19 – Search (I) procedure

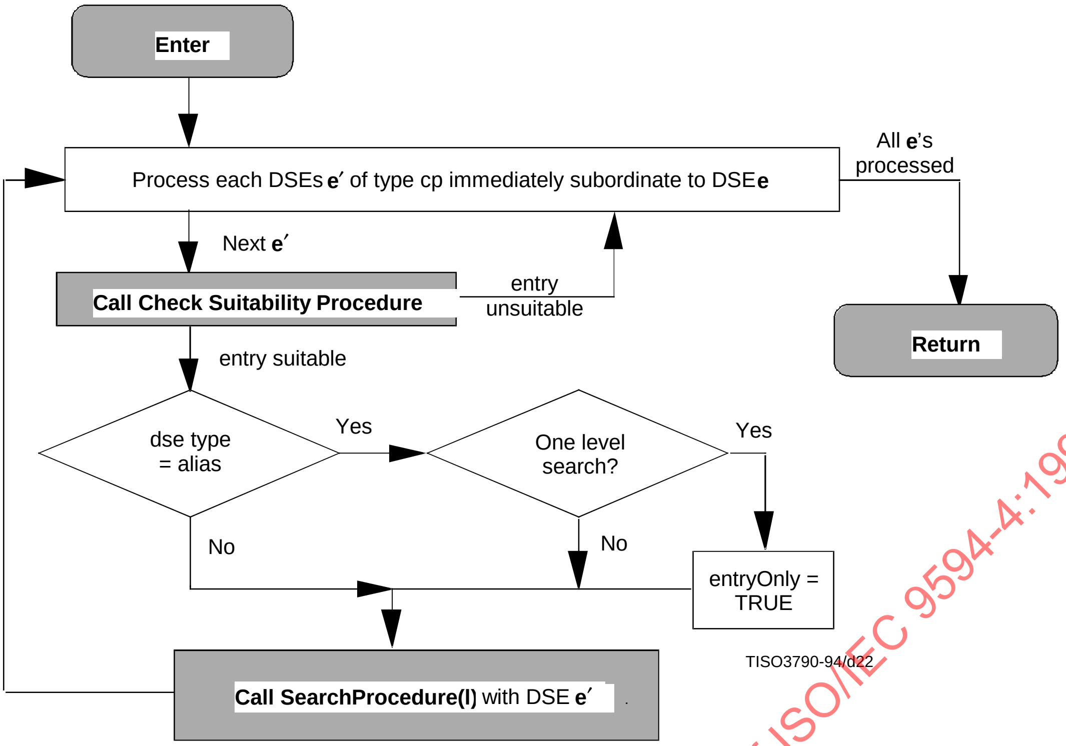


Figure 20 – Search (II) procedure

19.3.2.2.3 Search Alias procedure

This procedure is executed if a DSE of type **alias** has been encountered during the processing of a **search** request (see Figure 21):

1) If **subset** is **baseObject** or **oneLevel**, go to step 4).

2) If **aliasedEntryName** is a prefix of **targetObject** or **baseObject** or any of the previous values of the **targetObject** in **ChainingArguments.traceInformation**, then the alias is excluded from the Search because this would cause a recursive search with duplicate results.

3) If **targetObject** or **baseObject** or any of the previous values of the **targetObject** in **ChainingArguments.traceInformation** is a prefix of **aliasedEntryName**, then no specific processing of the alias is required because the aliased subtree will be searched anyway.

   NOTE – For both of the above cases, **baseObject** may not be prefix of **targetObject**, due to alias dereferencing.

4) Build a **DSP** request with the **targetObject** set to the **aliasedEntryName**. If **subset** is **oneLevel**, set **entryOnly** to **TRUE**. Call the Operation Dispatcher for the request to be locally continued.

5) If the **Operation Dispatcher** returns a referral error, or busy, or unavailable errors then add (or make and add) the continuation reference to **partialOutcomeQualifier** of **SearchResult**, and return.

6) If the **Operation Dispatcher** returns other errors, discard it and return.

7) If the **Operation Dispatcher** returns a **SearchResult**, then:

   i) If the result is signed, encrypted, or signed and encrypted, add it to **uncorrelatedSearchInfo** in **SearchResult**.

   ii) If the result is not signed, encrypted, or signed and encrypted, add it to **searchInfo** in **SearchResult**.

   And return.

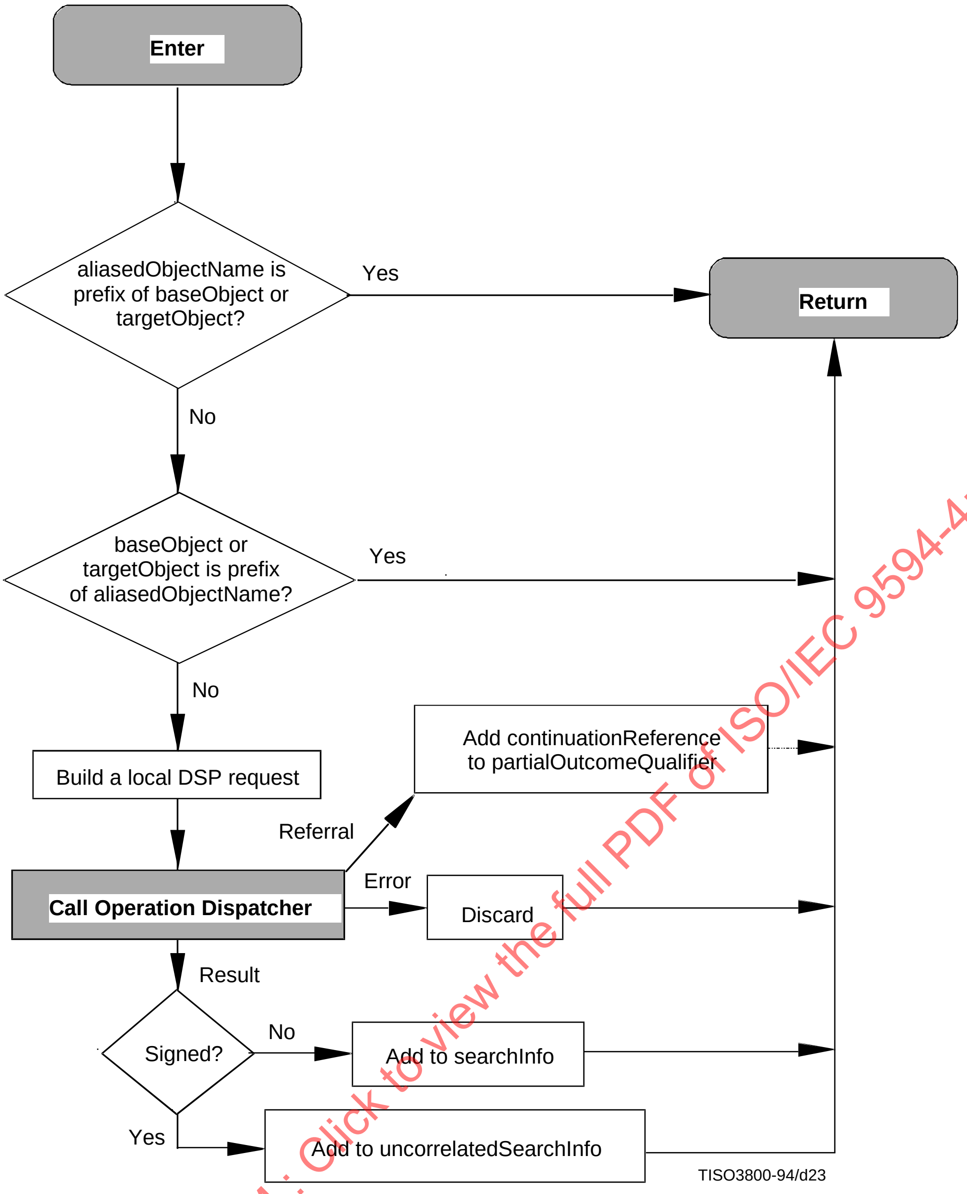


Figure 21 – Search alias procedure

20 Continuation Reference procedures

The procedures in this clause are called to process the list of continuation references (**NRcontinuationList** or **SRcontinuationList)** created by other procedures.

The **Continuation Reference** procedures consist of the steps shown in Figures 24, 25, and 26. The first stage is to identify sets of continuation references from the continuation list that have a common target object component. These have been created from a set of subordinate or non-specific subordinate references associated with the same entry in the DIT. Within each of these sets there may be continuation references which occur more than once. The sets should be scanned and any duplicates found should be discarded.

These sets (each with a different **targetObject** component) may be processed independently, either sequentially or in parallel by the DSA, since there is no risk that the same results will be returned from any two sets. However, the processing of each continuation reference within one set, and of each **AccessPointInformation** within one continuation reference, and of each access point within one **AccessPointInformation**, has to be controlled, or duplicate results may occur, as described in 20.1.

The procedure adopted in the **APInfo** procedure, is to process one by one the set of access points contained in a single **AccessPointInformation**. These all point to (copies of) the same naming context (or possibly a set of naming contexts held in one DSA, in the case of NSSRs). If the first access point produces a result or a hard error, further access points do not need to be processed. However, if the error is a soft error, i.e. a **serviceError** (with problem **busy**, **unavailable**, **unwillingToPerform**, **invalidReference,** or **administrativeLimitExceeded**), then the DSA may choose, as a local option, to process another access point from the set.

Processing of the **AccessPointInformation** values within one set of continuation references, is handled in a uniform way, irrespective of which continuation reference they originated from. (This is because two DSEs of type **subr** below a single entry would produce two continuation references, each containing one **AccessPointInformation** value, whereas one DSE of type **nssr** to the same two subordinates (assuming that they are held in different DSAs), would produce one continuation reference containing a set of two **AccessPointInformation** values.)

The **accessPointInformation** values may be processed either sequentially or in parallel, as described in 20.1. The parallel strategy is more likely to produce duplicate results. Duplicates shall always be discarded.

20.1 Chaining strategy in the presence of shadowing

In the presence of shadowing, a DSA may choose between different strategies when it has to multi-chain request to more than one DSA. This choice always occurs if the DSA has to process more than one continuation reference with the same **targetObject**. This situation can occur from multi-chaining caused by NSSR decomposition during Name Resolution (as shown in Figure 22) or from request decomposition during the evaluation of a multiple object operation (see Figure 23).

The goal of these strategies is to deal with the problem of duplicate results and duplicate processing when shadowed information is used in multi-chaining of requests (caused by either NSSR or request decomposition). For example, in Figure 22, DSA 1 multi-chains a request to both DSAs 2 and 3 because of the NSSR held in DSE B. If the use of shadowed information is allowed, both DSAs 2 and 3 may apply the chained operation to both subtrees starting at X and Y.

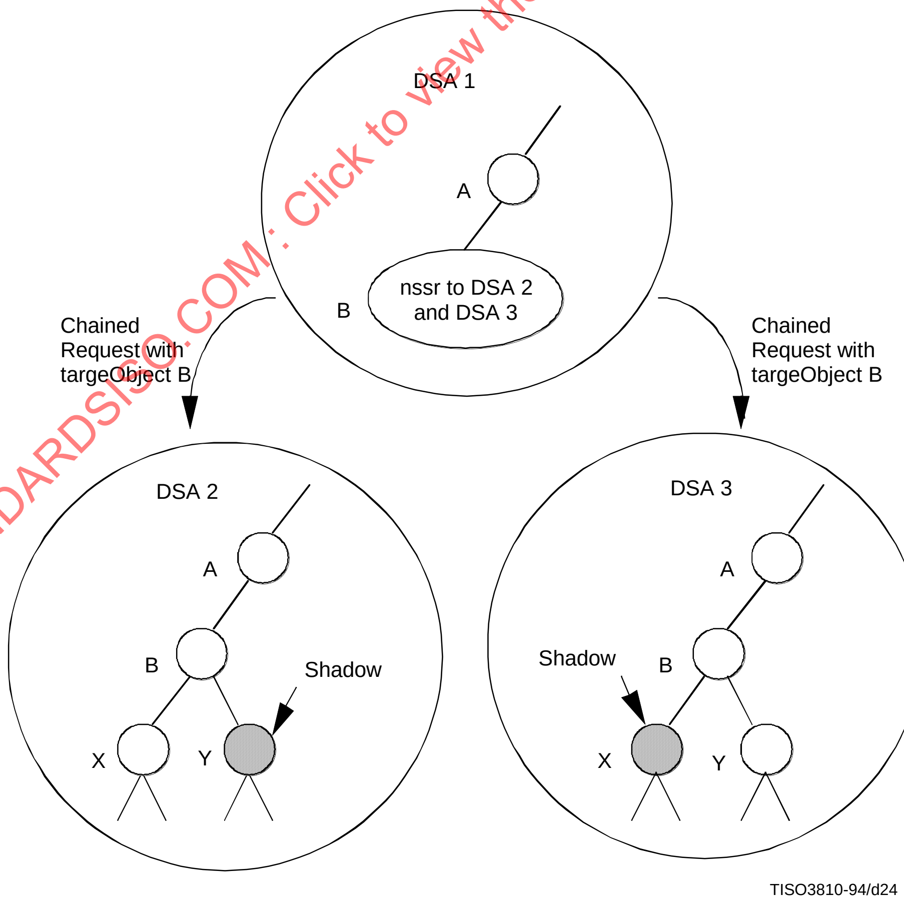


Figure 22 – Multi-chaining caused by NSSR during Name

Similarly, in Figure 23, DSA 1 multi-chains (as a result of request decomposition) to the two subordinate references held in DSEs X and Y. Again, if the use of shadowed information is allowed, both DSAs 2 and 3 may apply the chained operation to both subtrees starting at X and Y.

To deal with this problem of duplication, a DSA may choose one of the following strategies when multi-chaining to multiple DSA requests with the same **targetObject**.

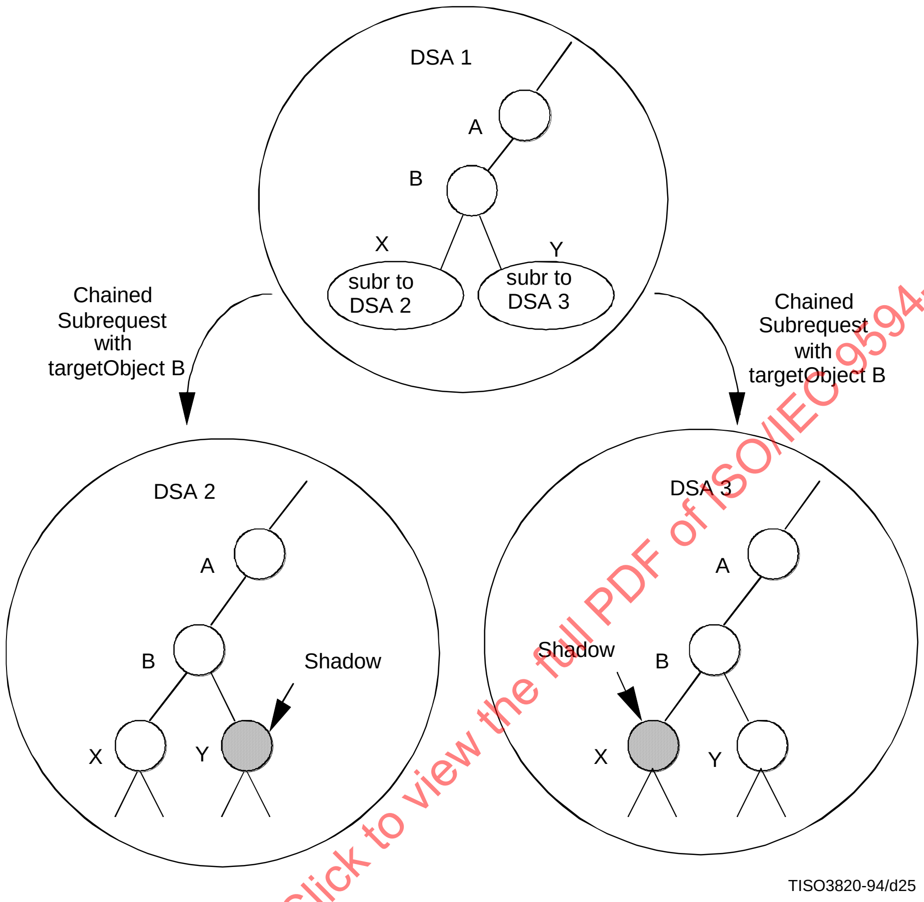


Figure 23 – Multi-chaining Request Decomposition using Subordinate

20.1.1 Master only strategy

A DSA may choose this strategy to prevent the usage of shadowed information when performing a parallel or sequential multi-chaining caused by NSSR decomposition, or request decomposition during a Search or List evaluation. For this strategy, during a Search or List operation evaluation the **excludeShadows** component of the **ChainingArguments** is set to **TRUE**. If NSSRs are encountered during Name Resolution, a DSA may set **nameResolveOnMaster** to **TRUE** to ensure that only a single path is followed. **nameResolveOnMaster** shall be set to **TRUE** if NSSR are encountered and the operation is one of the Directory modification operations. In either case, only the DSA(s) that hold the master entry (or entries) relevant to the operation shall perform the operation. This master only strategy can be used during both parallel as well as sequential multi-chaining.

NOTE – Setting **nameResolveOnMaster** to **TRUE** eliminates the possibility of multiple paths during name resolution by (1) ignoring shadow entries and (2) by ensuring that only one DSA may proceed with name resolution in situations where a complex DIT distribution would otherwise permit more than one to proceed. This is achieved by allowing only the DSA holding the master entry corresponding to the first **nextRDNToBeResolved** RDNs of the target object name to continue with name resolution. Any other DSAs will not be able to proceed even though they may hold master entries which match more of the target object name.

20.1.2 Parallel strategy

Using this strategy, a DSA sends out all chained requests by parallel multi-chaining. This strategy may be used during Search or List evaluation, and name resolution of the NSSRs. This will allow the use of shadowed information for processing of the chained requests, but may result in duplicate executions and duplicate results for the operation. If a DSA selects this strategy, it shall remove duplicate results from the operation result that it returns.

Because the removal of duplicate results is not possible if a signed result has been requested, a DSA shall not choose this strategy if signed results are requested during Search evaluation, unless **excludeShadows** is also set.

20.1.3 Sequential strategy

This strategy avoids duplicate results by using sequential multi-chaining to process the chained (sub)requests of a Search decomposition or of a NSSR decomposition. Each chained request is processed one after the other.

In the case of NSSR decomposition, if a result or a hard error is returned to a request, further requests do not need to be chained. If a soft error is returned, a further request may be chained, or the soft error returned to the requester, depending upon local policy.

In the case of Search evaluation, the **exclusions** component of the **ChainingArguments** is set to the set of RDNs that have already been processed. This is done by incorporating the elements in **ChainingResults.alreadySearched** to the **exclusions** argument of the next chained request. This is the only strategy that completely avoids duplication during Search evaluation.

A sequential strategy is not defined for List evaluation (although sequential multi-chaining may be used), since a superior DSA has no way of excluding specific subordinates from being returned in further List subrequests (note that **excludeShadows** does not exclude specific subordinates, but rather is a coarse way of excluding all shadows).

20.2 Issuing chained subrequests to a remote DSA

Prior to issuing a subrequest, a DSA has to execute a **dSABind** operation when the DSA has to establish an association to the remote DSA. Management of associations is outside the scope of the Directory Specifications. An association to another DSA is considered unavailable if the association cannot be established or the DSA for local reasons decides not to establish one. In this case, the **dSABind** has failed. It is a local decision when to stop trying to establish an association and declare an association as unavailable.

When a DSA tries **dSABind** to another DSA and receives a **directoryBindError**, the issuing of the subrequest failed.

20.3 Procedures' parameters

20.3.1 Arguments

These procedures make use of the following arguments:

- the list of continuation references to process in **NRcontinuationList (**for the **Name Resolution Continuation Reference** procedure), and **SRcontinuationList** (for the **List Continuation Reference** and **Search Continuation Reference** procedures, respectively);
- the **CommonArguments** of the operation argument;
- the **ChainingArguments**.

20.3.2 Results

These procedures create the following results:

- a list of received results/errors of issued chained requests if chaining has been selected;
- an updated list of unprocessed continuation references in **continuationList**.

20.3.3 Errors

These procedures can return one of the following errors:

- a **serviceError** with problem **outOfScope** in the case that a referral would have been created which is not within **scopeOfReferral**;
- a **serviceError** with problem **ditError** in the case that an invalid knowledge reference has been detected;
- a **nameError** with problem **noSuchObject** in the case that all subrequests from NSSR decomposition returned **unableToProceed**;
- any other error that is returned by a chained subrequest;
- a **referral** in the case that chaining was not selected and **operationProgress.nameResolutionPhase** is set to **notStarted** or **proceeding**.

20.4 Definition of the Procedures

If **operationProgress.nameResolutionPhase** is set to **notStarted** or **proceeding**, the procedure in 20.4.1 (**Name Resolution Continuation Reference** procedure) shall be followed. The multiple entry interrogation operations List and Search respectively call the procedures in 20.4.2 and 20.4.3.

20.4.1 Name Resolution Continuation Reference procedure

The **Name Resolution Continuation Reference** procedure consists of the steps as shown in Figure 24. The basic principle of this procedure is to sequentially process the set of continuation references created during Name Resolution. The following steps shall be executed for each continuation reference **C** contained in **NRcontinuationList** in a selected order until all references have been processed or an error or result has been returned. If all references have been processed, return to the **Operation Dispatcher** to continue with the **Result Merging** procedure to process the received result or referral.

1) Check whether **chainingProhibited** is set. If it is set, then the DSA is not allowed to chain. According to **local policy**, either a **serviceError** with problem **chainingRequired** or a **referral** is returned to the **Operation Dispatcher**.

2) If **chainingProhibited** is not set, then check if **local policy** allows chaining. If chaining is not allowed, then return a **referral**. If **local policy** allows chaining, then continue with the next step.

3) Process each of the Continuation References of the list of Continuation References found in **NRcontinuationList**. If there are no more unprocessed Continuation References then return with **serviceError**.

4) Process the next Continuation Referenced C from **NRcontinuationList**. If it is a NSSR, then continue at step 5). If it is not a NSSR, then call the **APInfo** procedure to process it. Distinguish between the possible returns of the **APInfo** procedure:

   – If the **APInfo** procedure returns a **null result**, continue at step 3) with processing the next Continuation Reference.

   – If the **APInfo** procedure returns an **error**, **referral** or **result**, then return it.

5) In this case, the Continuation Reference is of type NSSR and the DSA has the choice of doing sequential or parallel chaining, depending on the local choice of strategy. If the NSSR is to be processed sequentially, then continue at step 6). If it is to be processed in parallel, then for each of the **AccessPointInformation (API)** in the NSSR the **APInfo** procedure is called so that they are processed in parallel. Wait for all the API to be processed, i.e. wait for all the calls to the **APInfo** procedure to return. Check all the results received from the call to the **APInfo** procedure in the following order:

   – If all the calls return a **serviceError** with problem **unableToProceed** and **partialNameResolution** is **FALSE**, then return **nameError**.

   – If all the calls return a **serviceError** with problem **unableToProceed** and **partialNameResolution** is **TRUE**, then in the result set **partialName** to **TRUE**, **nameResolutionPhase** to **completed**, set **entry suitable** (this will be for the **lastEntryFound**), and go to the appropriate operation evaluation.

   – If one or more **results** are received, then **discard possible duplicates** and return the **result**.

   – If an **error** is received that is not a **serviceError** (e.g. a **nameError**), then return an **error**.

   – Otherwise return a **referral** or **serviceError** to the **Operation Dispatcher**, according to local choice.

6) Choose the next unprocessed **API** from the set of **API**s in the NSSR and continue at step 7). If all the **API**'s have been processed, then check if all the calls to the **APInfo** procedure returned a **serviceError** with problem **unableToProceed**.

   – If they did and **partialNameResolution** is **FALSE**, then the entry cannot be found and a **NameError** is returned. If they did and **partialnameResolution** is **TRUE**, then in the result set **partialName** to **TRUE**, **nameResolutionPhase** to **completed**, set **entry suitable** (this will be for the **lastEntryFound**), and go to the appropriate operation evaluation. If they did not, then according to local choice, return a **referral** or **serviceError**.

7) Call the **APInfo** procedure. Distinguish between the possible results from the call to **APInfo** procedure:

   – If a **serviceError** with problem **unableToProceed** is received, try another Access Point. Continue at step 6).

   – If a **serviceError** with problem **busy**, **unavailable**, **unwillingToPerform** or **invalidReference** is received, then the indicated problem may be of a transient nature and it is a local choice to try and chain the request on to another DSA. If it is chosen to try another DSA, then continue at step 6); otherwise return a **referral** or **serviceError**, according to local choice.

– If an error other than **serviceError** with problem **busy**, **unavailable**, **unwillingToPerform**, **invalidReference** or **unableToProceed** is received, that error should be returned to the **Operation Dispatcher**. If the **serviceError** is **invalidReference**, this shall be converted into **ditError** before being returned to the requester.

– If a **result** or **referral** is received, return it to the **Operation Dispatcher**.

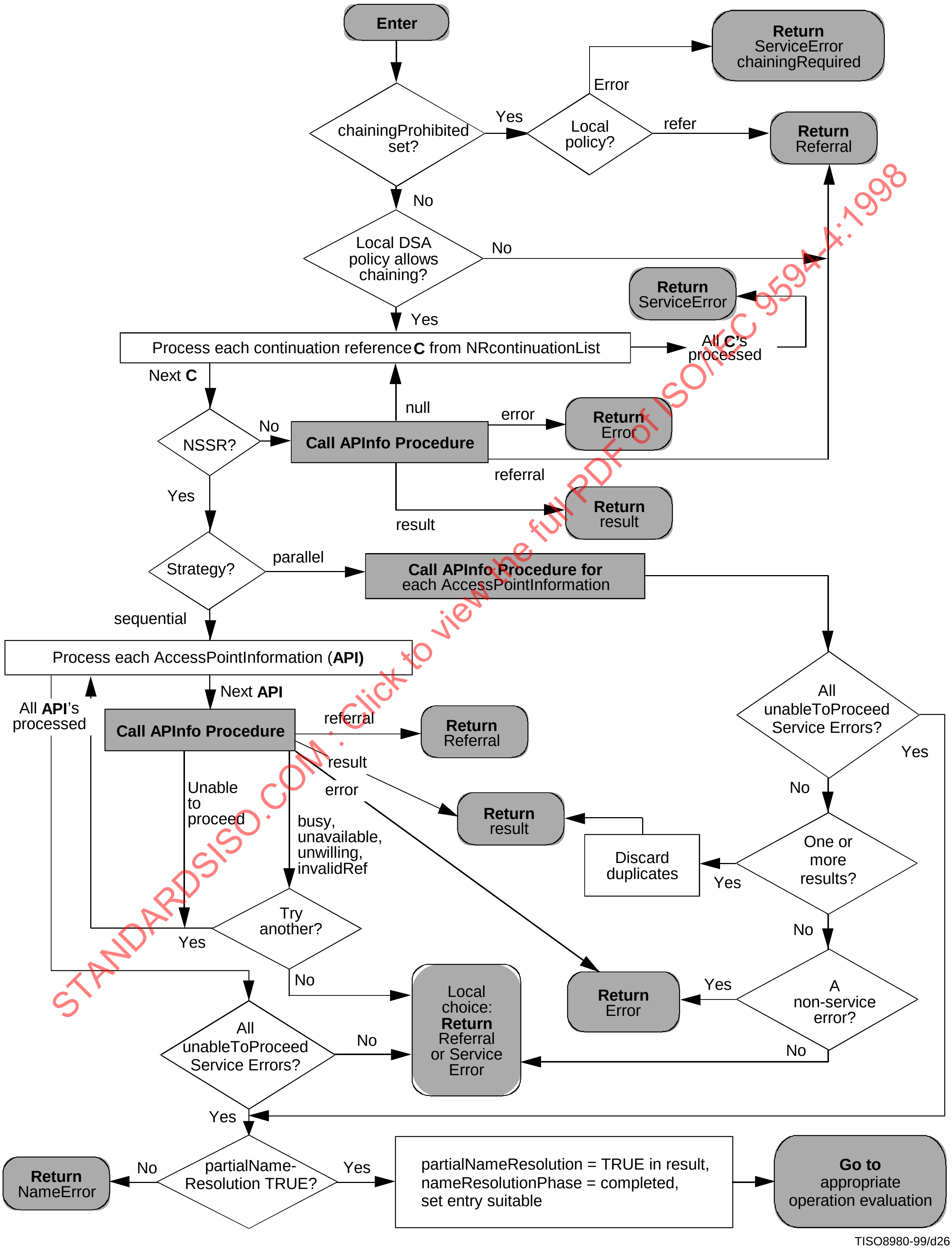


Figure 24 – Name Resolution Continuation Reference

20.4.3 List Continuation Reference procedure

The **List Continuation Reference** procedure consists of the steps shown in Figure 25. This procedure is invoked when a List request cannot be satisfied in the local DSA and a set of continuation references have been added to **SRcontinuationList** for chaining or referral. All these continuation references (CR) have the same **targetObject**. Those CRs with **referenceType nssr** have one or more **AccessPointInformation** values (**API**s), whereas other type CRs have only one **API** in them. Each of these **API** is extracted and considered for chaining or referral.

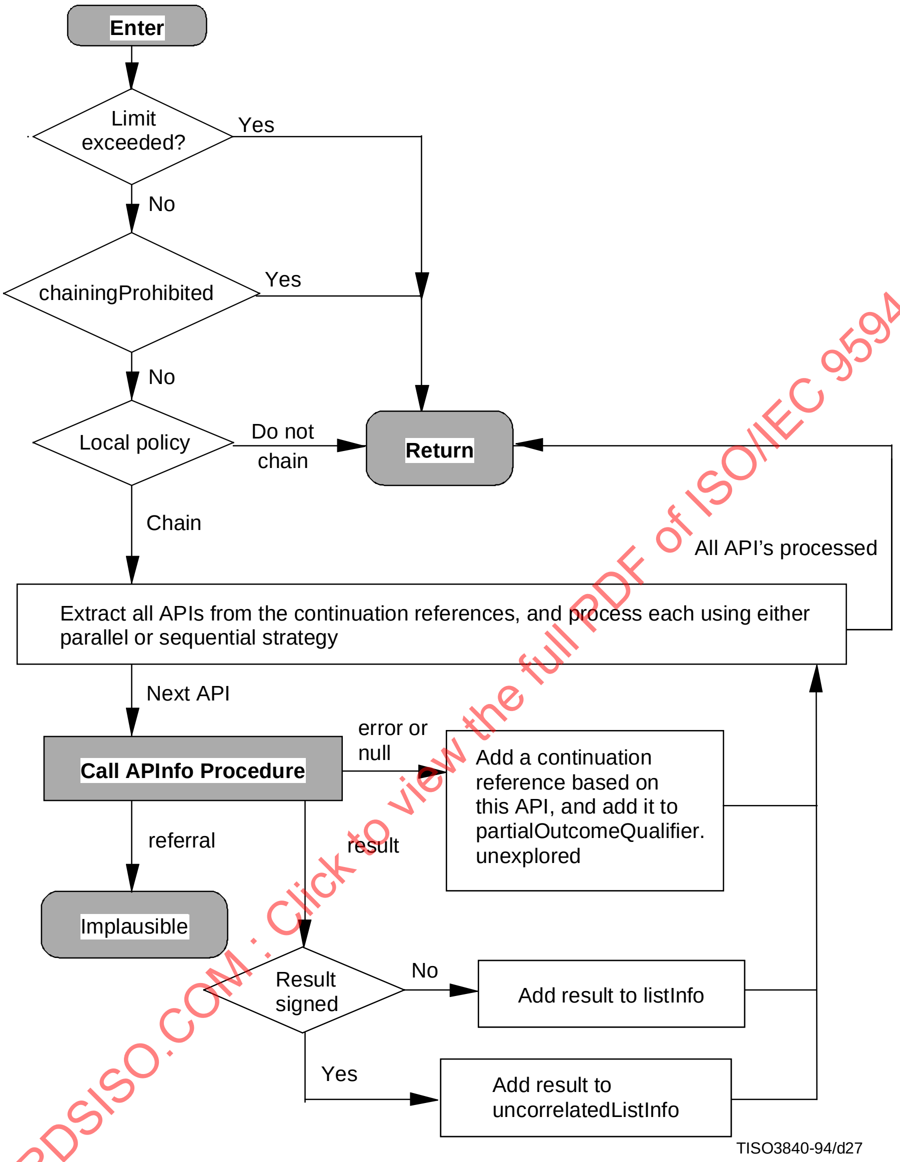


Figure 25 – List Continuation Reference procedure

The following steps shall be executed:

1) If any of the limit problem has been exceeded thus far, then return to the **Operation Dispatcher** to continue with the **Result Merging** procedure**.**

2) If the **chainingProhibited** flag in **CommonArguments.serviceControls** is set or the DSA decides not to do any chaining because of its local operational policy, then the DSA shall directly return to the **Operation Dispatcher** to continue with the **Result Merging** procedure.

3) Create a set of **AccessPointInformation** values from the **accessPoints** component of every continuation references in the **SRcontinuationList**.

   Use either parallel or sequential strategy to process each **API** as follows:

   i) Call the **APInfo** procedure with the next **API** in the set.

   ii) If a result is returned then add it to **listInfo** if it is not signed, or add it to **uncorrelatedListInfo** if it is signed.

iii) If the return is an error or null, it means that **APInfo** has already tried all access points in the **API** without success. Based on local operational and security policy, either ignore and proceed to the next **API**, or add a continuation reference based on this **API** to the **partialOutcomeQualifier**.

NOTE – It is not plausible to get a referral back from **APInfo**. Any "referral" should come in the form of **unexplored** in **partialOutcomeQualifier**.

4) When all **API**s are processed, return to the **Operation Dispatcher**.

20.4.4 Search Continuation Reference procedure

The **Search Continuation Reference** procedure consists of the steps shown in Figure 26. This procedure is invoked when a Search request cannot be satisfied in the local DSA and a set of continuation references have been added to **SRcontinuationList** for chaining or referral. The procedure is very similar to the **List Continuation Reference** procedure. The difference is that in this case the continuation references in **SRcontinuationList** may have different **targetObject** values. Thus, the continuation references are sorted into sets of continuation references with the same **targetObject**. Also, the use of **exclusions** in chaining arguments and of **alreadySearched** in chaining results is defined, as this is an important strategy for search. The use of **exclusions** and **alreadySearched** is applied to processing each set of continuation references with the same **targetObject**.

The following steps shall be executed:

1) If any of the limit problem has been exceeded thus far, then return to the **Operation Dispatcher** to continue with the **Result Merging** procedure.

2) If the **chainingProhibited** flag in **CommonArguments.serviceControls** is set or the DSA decides not to do any chaining because of its local operational policy, then the DSA shall directly return to the **Operation Dispatcher** to continue with the **Result Merging** procedure.

3) Sort the continuation references in **SRcontinuationList** into sets that have the same **targetObject**. Within each set, remove any duplicates.

NOTE 1 – If one or more **targetObject** values is not a primary RDN, then this sorting may not be accurate. The sorting shall take into account alternative distinguished RDNs, if known.

4) For each subset of continuation references, create a set of **AccessPointInformation** values from the **accessPoints** component of every continuation reference in the subset, and choose either sequential or parallel strategy for further processing. If the parallel strategy is chosen, then skip the steps below that are indicated only applicable to the sequential strategy.

a) If the sequential strategy is chosen, maintain a local variable **localExclusions** for each set of continuation references that have the same **targetObject**. Initially, **localExclusions** is set to the **exclusions** of the incoming chaining request (if it exists), and all locally searched subtrees directly under **targetObject**.

b) If the sequential strategy is used, compare the **targetObject** to all the elements of **localExclusions**, and remove those elements which do not contain **targetObject** as a prefix. These are the relevant exclusions for the current target object.

c) Extract all the **API**s from all the continuation references of the current target object's set.

d) Loop through each **API**. For each **API**:

i) Call **APInfo**.

ii) If a result is returned, then add the result to **searchInfo** if it is not signed, or add it to **uncorrelatedSearchInfo** if it is signed. If the sequential strategy is used, update **localExclusions** using **alreadySearched** in the incoming reply, and also merge the **alreadySearched** in the incoming reply to this DSA's **ChainingResults.alreadySearched**. Then proceed to the next **API**.

iii) If an error or null is returned, it means that **APInfo** has already tried all access points in the API without success. Based on local operational and security policy, either ignore and proceed to the next **API**, or add a continuation reference based on this **API** to the **partialOutcomeQualifier**.

NOTE 2 – It is not plausible to get a referral back from **APInfo**. Any "referral" should come in the form of **unexplored** in **partialOutcomeQualifier**.

e) When all **API**s are processed, proceed to the next set of continuation references with the same **targetObject**.

5) When all the continuation references are processed, return to the **Operation Dispatcher**.

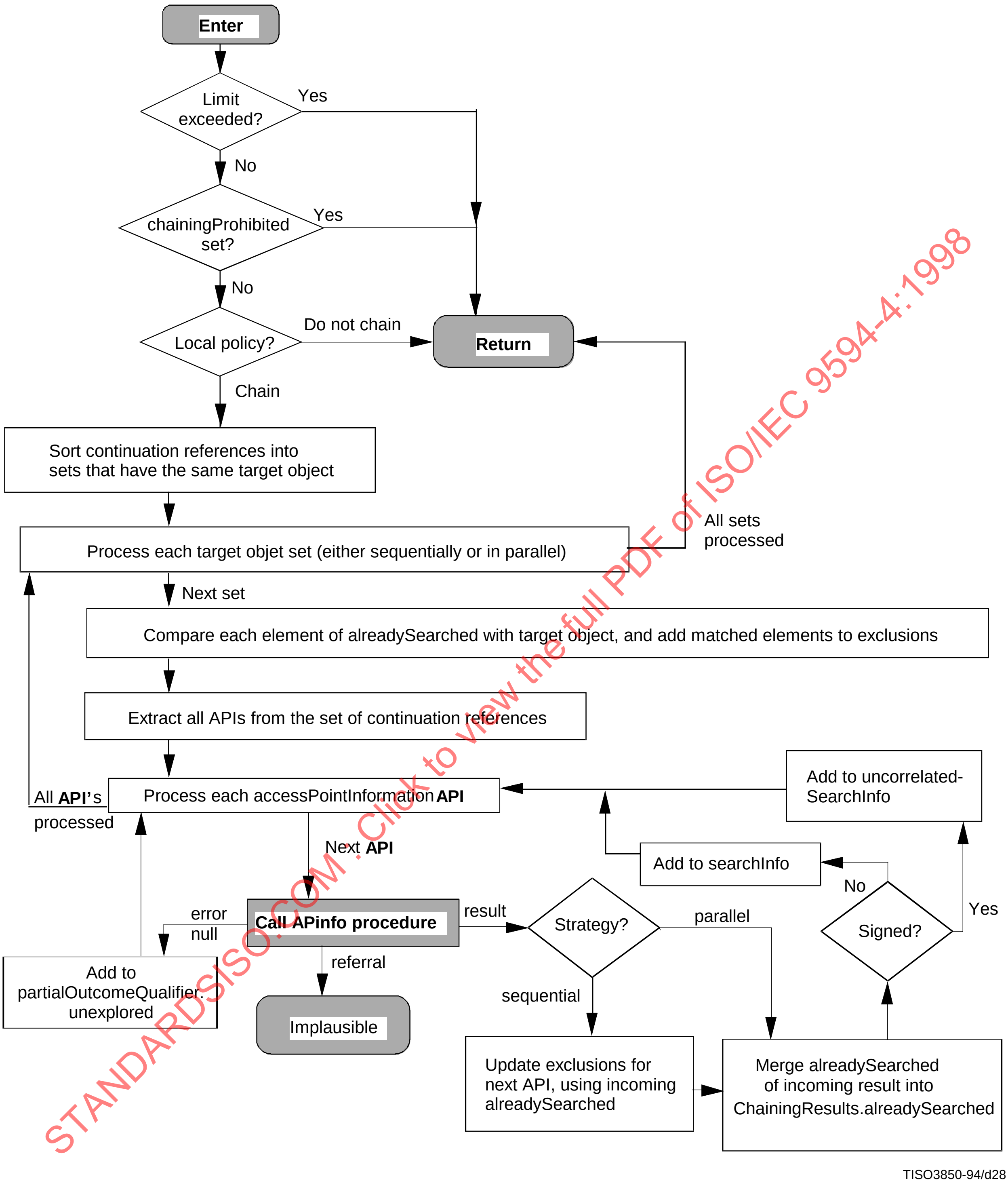


Figure 26 – Search Continuation Reference

20.4.5 APInfo procedure

This procedure is called to process an **AccessPointInformation**, which contains one or more access points (see Figure 27). They are processed one by one until either a result or error is returned. If the error is a service error such that trying another access point may succeed, then additional access points are tried as long as local operational policy permits:

1) Perform loop detection. If a loop is detected, return **serviceError** with problem **loopDetected**. Otherwise, continue at step 2).

2) Process each of the access points from the access point information. If all have been processed, return a **null result**. If there is any access point to process, continue at step 3).

3) Check whether local policy allows chaining to this access point. This check should take into account the settings of the service controls and chaining arguments (e.g. **chainingProhibited**, **preferChaining**, whether the access point is within the **localScope** or not, **excludeShadows**). If the local policy or the setting of the respective service controls do not allow the use of this particular access point, then ignore the access point and continue at step 2). If the access point can be used, continue at step 4).

4) If local policy selected the master only strategy, then set the chaining argument **excludeShadows** to **TRUE**.

   If **nameResolutionPhase** is not **completed** and the strategy is to continue name resolution on master entries, then set **nameResolveOnMaster** to **TRUE**.

   The chaining argument **nameResolveOnMaster** shall be set to **TRUE** if either of the following is true:

   – in the incoming chaining argument **nameResolutionPhase** is **proceeding** and **nameResolveOnMaster** is **TRUE**; or

   – the operation is one of the modification operations, the **referenceType** of the chaining request to be issued is NSSR, and a parallel strategy is used.

   NOTE – This method of using **nameResolveOnMaster** is to prevent modification operations being applied multiple times due to the presence of NSSR.

5) Build a chained request and try to issue it:

   a) Perform loop avoidance by checking if an item with the same **targetObject** and **operationProgress** occurs in **traceInformation** of the received **ChainingArguments**. If the resulting request (as described in step 5), c) would result in a loop, then the DSA shall either return a **serviceError** with problem **loopDetected** to the requesting DUA/DSA or ignore the access point and try the next access point by continuing at step 2).

   b) If the request or subrequest to be chained is the result of executing a referral, then an extra check for loop avoidance is required. Check if an item with the same **targetObject**, **operationProgress** and target DSA occurs in **referralRequests**. If so, then take the action specified in a). If not, then add a new **TraceItem** to **referralRequests** with the following components:

      – **targetObject** and **operationProgress** set to the value of the chained request/subrequest;

      – **dsa** set to the name of the DSA to which the request/subrequest is to be chained.

   c) After a successful Bind, the DSA shall issue a chained operation of the same operation type as the operation that is processed with the following parameters:

      – the operation argument within the chained operation is set as for the operation argument received;

      – **ChainingArguments.originator** set as received;

      – **ChainingArguments.targetObject** set to the **targetObject** of the continuation reference;

      – **ChainingArguments.operationProgress** set to the value of **operationProgress** of the continuation reference;

      – **ChainingArguments.traceInformation** set to trace information as updated by the **Request Validation** procedure;

      – **ChainingArguments.aliasDereferenced** to the updated value of the locally updated **aliasDereferenced**;

– **ChainingArguments.returnCrossRefs** to a local choice;

– **ChainingArguments.referenceType** to the value of **referenceType** of the continuation reference;

– **ChainingArguments.timeLimit** to the value of the received **timeLimit**;

– **ChainingArguments.exclusions** absent;

– **SecurityParameters** set to the value of the received **SecurityParameters**.

6) If the request could not be issued successfully, then continue at step 7). If it could be issued successfully continue at step 8).

7) It is a local choice whether or not to continue. If the DSA chooses to continue, then the error is ignored and the next access point will be tried. Continue at step 2). If the DSA decides not to try another access point, then it is a choice of local policy whether to return a respective **referral** or a **serviceError** to the caller of the procedure.

8) If the request could be issued successfully, then the DSA shall wait for the reply and process it:

a) If a **result** is received, the **result** is returned to the caller of the procedure.

b) If a **serviceError** with problem **busy**, **unavailable**, **unwillingToPerform** or **invalidReference** is received, continue at step 7).

c) If **referral** is received and **returnToDUA** is set to **TRUE**, then the receiving DSA shall not act on the Referral, but shall return the Referral to the requester.

d) If a **referral** is received and **returnToDUA** is set to **FALSE**, then the same local policy considerations apply as in step 3) (taking into account service controls, chaining arguments, chaining strategy, etc.). If it is decided not to dereference the **referral**, then return the **referral** to the caller. If it is decided to dereference the **referral**, then empty the **NRcontinuationList**, place the Continuation Reference as received in the Referral in **NRcontinuationList** and call the **Name Resolution Continuation Reference** procedure. This may produce a **result**, **referral**, **serviceError** or other **error**. Whatever is received from the call of the **Name Resolution Continuation Reference** procedure shall be given back to the caller.

e) If any other **error** occurs, it shall be given back to the caller.

20.5 Abandon procedure

This procedure is invoked if an abandon request is received. It consists of the following steps as shown in Figure 28:

1) When an **abandon** request is received, which references an unknown operation, an **abandonFailed** with problem **noSuchOperation** shall be returned to the requester.

2) If the request to be abandoned has already been replied to, and the DSA has retained information to know so, an **abandonError** with a **tooLate** error value may be returned to the requester.

3) If the Abandon request is not valid, i.e. asks to abandon a request that is not an interrogation request, an **abandonFailed** with a **cannotAbandon** error value shall be returned to the requester.

4) If a DSA has outstanding chained (sub)requests when receiving a valid Abandon request for the original request, and the DSA decides to attempt abandoning, it may send Abandon requests for none, some, or all outstanding (sub)requests for the operation in question, and then wait for the replies to Abandon request and the outstanding (sub)requests. At any time during this operation the DSA may send an Abandon result and an **abandonFailed** to the requester and then discard replies to the issued Abandon requests and the outstanding (sub)requests as they arrive.

If the DSA decides not to send replies to the requester until there are no more outstanding (sub)requests, it may optionally send an **abandonedFailed** error to the requester if all the issued Abandon requests were replied to with **abandonedFailed** errors and if no local abandon operation has been performed.

If an **AbandonedFailed** error is returned to the requester, the original request shall be treated as if the Abandon request had never been received.

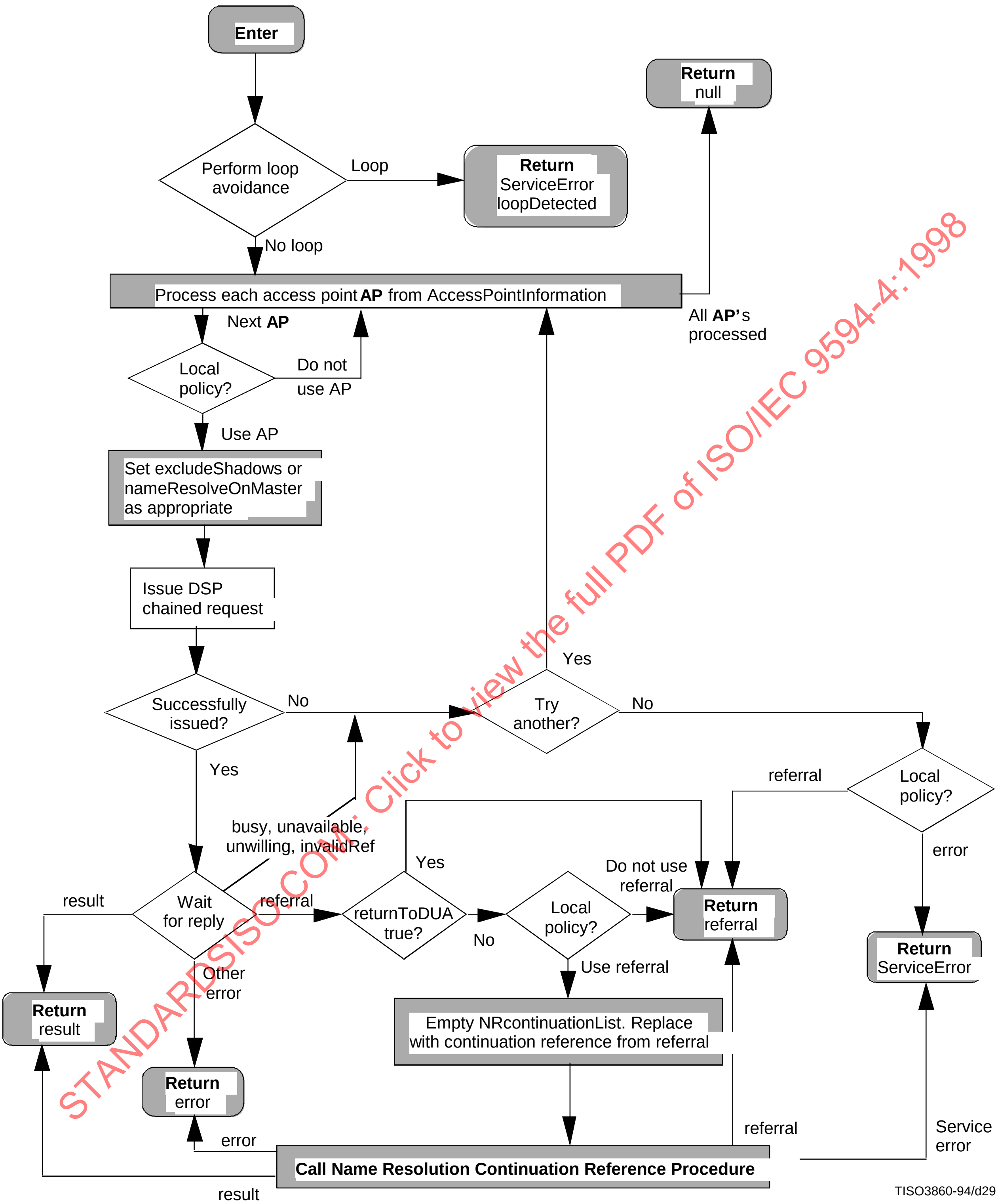


Figure 27 – APInfo procedure

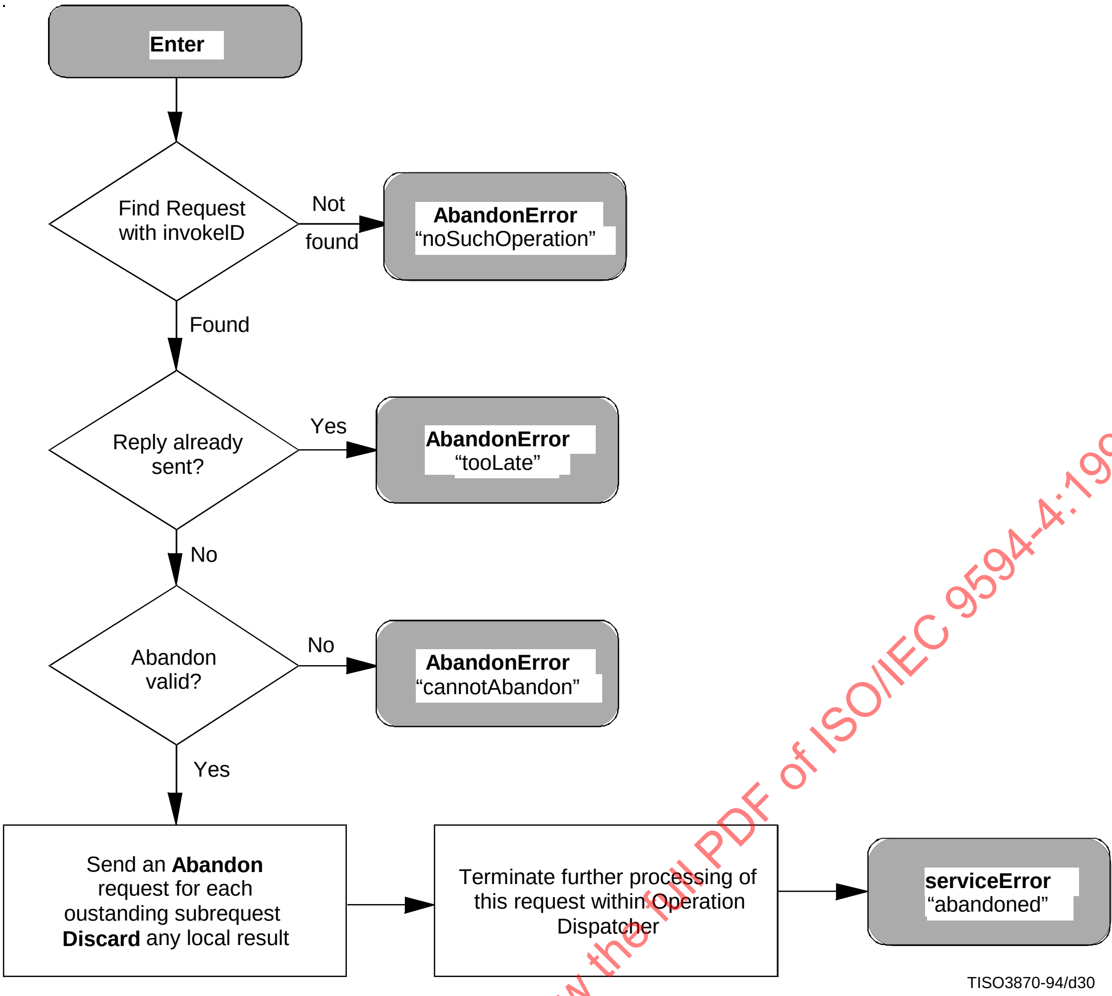


Figure 28 – Abandon

21 Results Merging procedure

The **Result Merging** procedure in Figure 29 is called following one of the **Continuation Reference** procedures. This procedure removes duplicates, if the result is not signed, and if there are additional continuation references in **partialOutcomeQualifier.unexplored**. Then the relevant Continuation Reference procedure(s) is called, if local operational policy permits:

1) If the operation is a List operation, continue at step 2); if the operation is a Search operation, then continue at step 3); otherwise, return the result that was supplied as input parameter to the **Result Merging** procedure.

2) The operation is a List operation. Remove all duplicates, giving preference to master information over shadow information.

   If the operation result was generated locally and it contains Continuation References then these will not be used for chaining but returned to the user. In this case, continue at step 6).

   If the operation result was received as the result of a Chained List operation, then the result might contain Continuation References. In this case, check if the **preferChaining** service control was set. If **TRUE**, the Continuation References should be used for chaining by the DSA. Continue at step 4).

3) The operation is a Search operation. Remove all duplicates, giving preference to master information over shadow information. If there is a limit problem then return the result. Otherwise continue at step 4).

4) Process each Continuation Reference that is in the **partialOutcomeQualifier.unexplored** of the result of any chained operation. If the local policy decides not to use it for chaining, then ignore it and choose another Continuation Reference. If the local policy allows the use of the Continuation Reference for chaining, then perform the following:

Check **nameResolutionPhase** that is supplied in the Continuation Reference. If it is **notStarted** or **proceeding**, then add it to the list of Continuation References that will be supplied to the **Name Resolution Continuation** procedure (**NRcontinuationList**). If **nameResolutionPhase** is **completed** then add the Continuation Reference to the list of Continuation References that is supplied to the subrequest Continuation procedure (**SRcontinuationList**).

Proceed until all Continuation References have been processed.

5) If there are Continuation References to be processed in **SRcontinuationList**, check the operation type. If the operation is a List operation, call the **List Continuation Reference** procedure and continue at step 2). If the operation is a Search operation, call the **Search Continuation Reference** procedure and continue at step 3).

   If **SRcontinuationList** is empty, then check if there are Continuation References in **NRcontinuationList**. If so, call the **Name Resolution Continuation Reference** procedure and continue at step 3).

   If both continuation lists are empty, continue at step 6).

6) Check whether the result is empty. If it is not empty then return it. If it is empty, either return a **null result** if the access control and local policy allows, or return an appropriate error.

If protection is performed, the merging of results shall not be performed.

22 Procedures for distributed authentication

This clause specifies the procedures necessary to support the directory distributed authentication services. These services, and hence the procedures, are categorized as:

- originator authentication, which is supported in either an unprotected (simple identity based) or secure (based upon digital signatures) form; and
- results authentication which is similarly protected (again based upon digital signatures).

22.1 Originator authentication

22.1.1 Identity-based authentication

The identity-based authentication service enables DSAs to authenticate the original requester of information for the purpose of effecting local access controls. DSAs wishing to exploit this service shall adopt the following procedure:

- For a DSA requiring to authenticate a DAP request, the DSA acquires the distinguished name of the requester through the Bind procedures at the time a DUA association (DUA to DSA) is established. Successful conclusion of these procedures does not in any way prejudice the level of authentication that may subsequently be required for processing operations using that association.
- The DSA with which the DUA association exists shall insert the requester's distinguished name in the initiator field of the **ChainingArguments** for all subsequent chained operations to other DSAs.
- A DSA, on receiving a chained operation, may satisfy that operation, or not, depending upon the determination of access rights (a locally defined mechanism). If the outcome is not satisfactory, a **securityError** may be returned with problem **insufficientAccessRights**.

22.1.2 Signature-based originator authentication

This signature-based originator authentication service enables a DSA to authenticate (in a secure manner) the originator of a particular service request. The procedures to be effected by a DSA in realizing this service are described in this clause.

The signature-based authentication service is invoked by a DUA using the **PROTECTED** variant of an optionally protected service request with **DirQOP signed** or **signedAndEncrypted**.

A DSA, on receiving a signed request from another DSA, shall remove that DSA's signature prior to processing the operation. Assuming the result of any signature verification proves to be satisfactory, the DSA will continue to progress the operation. If, during processing, the DSA needs to perform chaining, the argument set for each associated chained operation shall be constructed as follows:

- the DSA forms an argument set which may be optionally signed; the argument set comprises the incoming signed argument set together with a modified **ChainingArguments**.

In the event that the DSA is able to contribute information to the response, originator authentication, based upon the signed service request, may be used for the determination of access rights to that information.

If a DSA receives an unsigned service request for information which will only be released subject to originator authentication, a **securityError** will be returned with **SecurityProblem** set to **protectionRequired**.

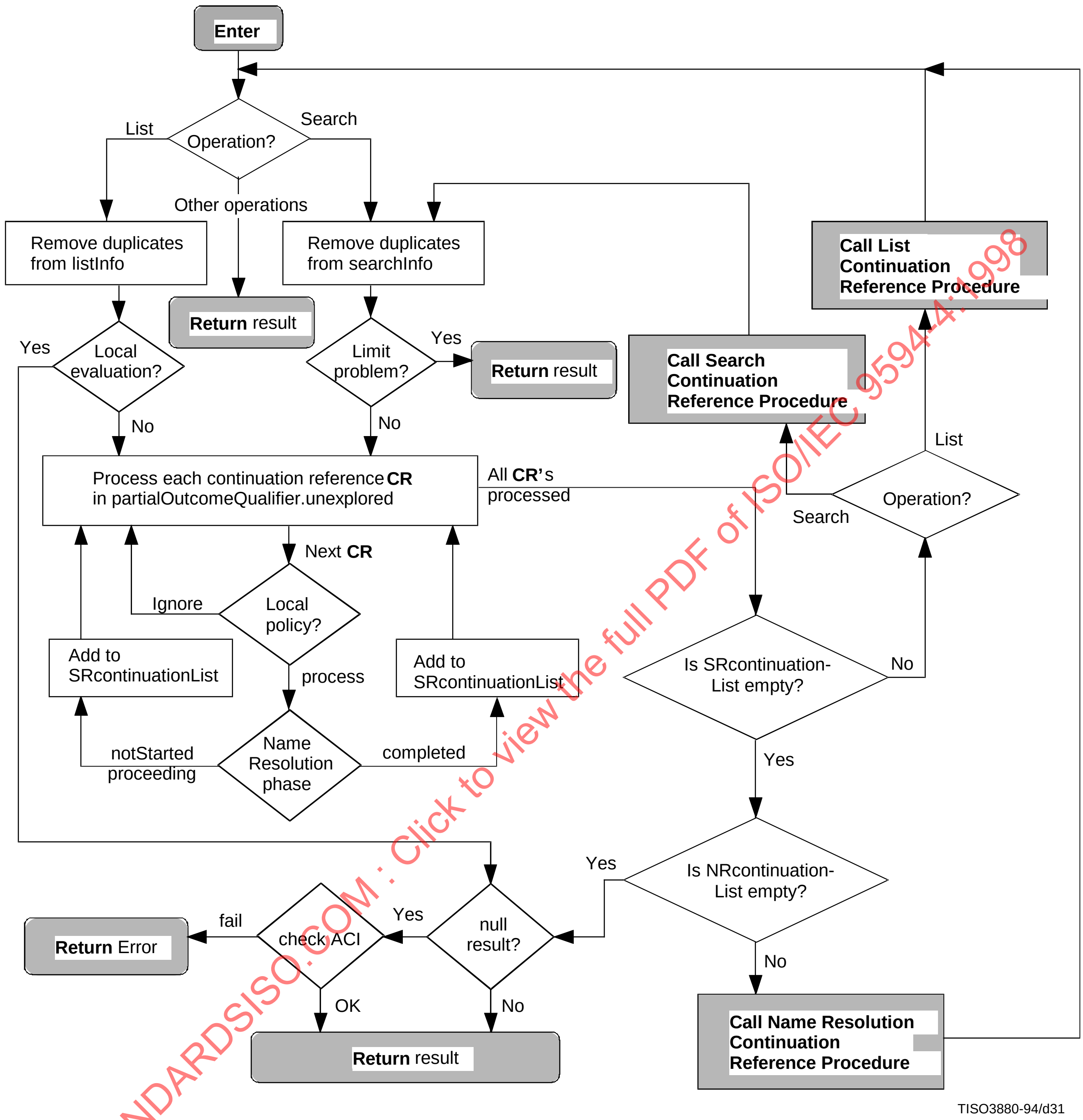


Figure 29 – Results Merging procedure

22.2 Results authentication

This service is provided to enable requesters of directory operations (either DUAs or DSAs) to verify (in a secure manner using digital signature techniques) the source of results. The results authentication service may be requested irrespective of whether originator authentication is to be used.

The results authentication service is initiated using the signed value of the **protectionRequest** component as contained within the argument set of directory operations; a DSA receiving an operation with this option selected may then optionally sign any subsequent results. The signed option in the protection request serves as an indication, to the DSA, of the requester's preference; the DSA may, or may not, actually sign any subsequent results.

In the case where a DSA performs chaining, the DSA has a number of options in terms of the form of results sent back to the requester, namely:

a) return a composite response (signed or unsigned) to the requester;

b) return a set of two or more uncollated partial responses (signed or unsigned) to the requester; within this set zero or more members may be signed and zero or one unsigned. In the event that an unsigned partial result is present, this member may in fact be a collation of one or more unsigned partial responses which have been received from other DSAs, contributed by this DSA, or both.

SECTION 6 – KNOWLEDGE ADMINISTRATION

23 Knowledge administration overview

To operate a widely distributed Directory with an acceptable degree of consistency and performance, procedures are required to create, maintain, and extend the knowledge held by each DSA. The following mechanisms together are used to administer a DSA's knowledge.

a) *Hierarchical* and *non-specific hierarchical operational bindings* – These procedures and protocols are defined in clauses 24 and 25. They are used to create and maintain subordinate references, non-specific subordinate references, and immediate superior references, as well as the context prefix information for naming contexts. These operational bindings are established between master DSAs holding naming contexts that are hierarchically related to each other as immediate subordinate to immediate superior. The procedures may be triggered as a side effect of modifying the RDN of, or adding or removing an entry, whose immediate superior is not held in the same DSA that holds the entry.

b) *Shadowing operational bindings* – These procedures and protocols are defined in ITU-T Rec. X.525 | ISO/IEC 9594-9. They are used to create and maintain knowledge references in two ways. First, as a side effect of establishing (or terminating) shadowing agreements, access points are added (or removed) from the **consumerKnowledge** and optionally the **secondaryShadow** operational attributes. This information may then be used by the procedures and protocols discussed above to update the subordinate reference in the superior master DSA and the immediate superior reference in the subordinate master DSA. Second, the DISP propagates the knowledge references held by master DSAs to shadow consumer DSAs.

c) *Cross references* – Cross reference distribution is a feature of the DSP. Its use to create and maintain cross references is summarized in 23.2.

NOTE – Mechanisms for initializing and maintaining the superior reference and **myAccessPoint** are outside the scope of this Directory Specification.

23.1 Maintenance of Knowledge References

This subclause describes how the DOP is used to maintain DSA operational attributes that express knowledge. A simple example of the relationship between knowledge attributes and the protocols employed to maintain them is described in Annex E.

23.1.1 Maintenance of consumer knowledge by supplier and master DSAs

A consumer reference is expressed through a value of the **consumerKnowledge** attribute, held by a shadow supplier DSA and associated with the context prefix for a naming context; a supplier reference, through a value of the **supplierKnowledge** attribute, held by a shadow consumer DSA and also associated with the context prefix for a naming context. Both attributes are held in DSEs of type **cp**. A value of each one of these attributes is created on establishment of the Shadow Operational Binding, and updated on modification of the Shadow Operational Binding.

A supplier DSA may obtain the information to construct values of the **secondaryShadows** attribute if the optional **secondaryShadows** component of its **ShadowingAgreementInfo** with a consumer is **TRUE**. In this case, whenever the consumer DSA detects that the set of DSAs holding copies of the commonly usable replicated area (its consumers, or, in turn, consumers of its consumers, etc., to whatever depth secondary shadowing might be carried) has changed (by addition, modification or deletion of access points), it communicates this new information (a set-of **SupplierAndConsumers**) by means of a **modifyOperationalBinding** operation, as described in ITU-T Rec. X.525 | ISO/IEC 9594-9.

A supplier DSA maintains its own **secondaryShadows** attribute associated with the context prefix as follows:

a) The set of **SupplierAndConsumers** received from a consumer by means of a **modifyOperationalBinding** operation may be used to create, or replace values of the attribute. The supplier component of **SupplierAndConsumers** represents the access point of a consumer DSA (or of its consumers, etc. depending upon the depth of secondary shadowing); the consumers component, the set of the consumer's consumers (or of their consumers, etc. depending upon the depth of secondary shadowing).

b) Every consumer providing its supplier with a **modifyOperationalBinding** operation containing a set of **SupplierAndConsumers**, includes the following values: the values of its **secondaryShadows** attribute, and a newly constructed value. This value is constructed using its own access point, **myAccessPoint**, (as the supplier component), and the values of the consumers' access points, contained within the **consumerKnowledge** attribute, that represent consumers holding commonly usable shadows (as the consumers component).

Recursive use of this procedure permits a master DSA for a naming context to know about all of its secondary shadow consumer DSAs holding commonly usable replicated areas derived from the naming context. This information is then available for the maintenance of subordinate, non-specific subordinate, and immediate superior references.

23.1.2 Maintenance of subordinate and immediate superior knowledge in master DSAs

A subordinate reference is expressed through a value of the **specificKnowledge** attribute, held in a DSE of type **subr** by the DSA holding the immediately superior naming context to that referenced; an immediate superior reference, through a value of the **specificKnowledge** attribute, held in a DSE of type **immSupr** by the DSA holding the immediately subordinate naming context to that referenced. A value of each one of these attributes is created in the superior and subordinate master DSAs on establishment of the HOB, and updated on modification of the HOB.

A subordinate master DSA provides a superior master DSA the information to construct its subordinate reference via the **accessPoints** component of the **SubordinateToSuperior** parameter it transfers to the superior in the DOP. The information included in **accessPoints** is determined by values of attributes held by the subordinate DSA as follows:

a) The value of the **myAccessPoint** attribute (held in the root DSE) is used to form the element in **accessPoints** with **category** having the value **master**.

b) The values of the **consumerKnowledge** and **secondaryShadows** (both held in the subordinate context prefix DSE) are used to form additional elements in **accessPoints** with **category** having the value **shadow**.

A superior master DSA provides a subordinate master DSA the information to construct its immediate superior reference via the **contextPrefixInfo** component of the **SuperiorToSubordinate** parameter it transfers to the subordinate in the DOP. This component is a value of type **SEQUENCE OF Vertex**, containing sequence of elements corresponding to the path from the root of the DIT to the subordinate context prefix. For one of these elements, corresponding to the context prefix of the immediately superior naming context, the optional component **accessPoints** will be present. The subordinate DSA holds this information as a **specificKnowledge** attribute in the DSE, of type **immSupr**, corresponding to this element of **contextPrefixInfo**. The information included in **accessPoints** by the superior DSA is determined by values of attributes held by the superior DSA as follows:

a) The value of the **myAccessPoint** attribute (held in the root DSE) is used to form the element in **accessPoints** with **category** having the value **master**.

b) The values of the **consumerKnowledge** and **secondaryShadows** (both held in the superior context prefix DSE) are used to form additional elements in **accessPoints** with **category** having the value **shadow**.

NOTE – Only those access points corresponding to consumer DSAs receiving commonly usable replicated areas should be selected by the superior and subordinate DSAs from their **consumerKnowledge** attributes for inclusion in **accessPoints**. The procedures for the construction of **secondaryShadows** guarantee that these access points will identify shadow DSAs holding commonly usable replicated areas.

23.1.3 Maintenance of subordinate and immediate superior knowledge in consumer DSAs

A shadow consumer DSA contracting with its supplier to receive the immediate superior and subordinate knowledge associated with a unit of replication, in effect, contracts to have its immediate superior and subordinate references maintained by its shadow supplier DSA via the DISP.

NOTE – For certain units of replication specifications, it may be necessary for the consumer DSA to contract to receive **extendedKnowledge** in order that subordinate knowledge may be provided to it by its supplier.

23.2 Requesting cross reference

To improve the performance of the Directory System, the local set of cross references can be expanded using ordinary Directory operations. If a DSA supports the DSP, it may request another DSA (which must also support the DSP) to return those knowledge references which contain information about the location of naming contexts related to the target object name of an ordinary Directory operation.

If the **returnCrossRefs** component of the **ChainingArguments** is set to **TRUE**, the **crossReferences** component of the **ChainingResults** may be present, consisting of a sequence of cross reference items.

If a DSA is not able to chain a request to the next DSA, a referral is returned to the originating DSA. If the **returnCrossRefs** component of **ChainingArguments** was **TRUE**, the referral may contain additionally the context prefix of the naming context which the referral refers to. The **contextPrefix** component is absent if the referral is based on a non-specific subordinate reference. The cross reference returned by a referral is based on knowledge held by the DSA which generated the referral.

In both cases (chaining result and referral) an administrative authority, through its DSA, may elect to ignore the request for returning cross references.

23.3 Knowledge inconsistencies

The Directory has to support consistency-checking mechanisms to guarantee a certain degree of knowledge consistency.

NOTE – In certain circumstances a knowledge reference will be accurate (not invalid in the senses described below) but not valid for use by a DSA because the DMD of the referenced DSA does not wish it to be contacted at all by the referencing DSA (e.g. a DSA which has somehow acquired a cross reference to the referenced DSA) or does not wish it to be contacted in a particular role (e.g. as the master DSA for a naming context).

23.3.1 Detection of knowledge inconsistencies

The kind of inconsistency and its detection varies for the different types of knowledge references:

a) *Cross and Subordinate references* – This type of reference is invalid if the referenced DSA does not hold a naming context or a replicated area derived from the naming context with the context prefix contained in the reference. This inconsistency will be detected during the Name Resolution process by inspection of the **operationProgress** and **referenceType** components of **ChainingArguments**.

b) *Non-specific Subordinate references* – This type of reference is invalid if the referenced DSA does not hold a local naming context with the context prefix contained in the reference minus the last RDN. The consistency check is applied as above.

c) *Superior References* – An invalid superior reference is one which does not form part of a reference path to the root. The maintenance of superior references shall be done by external means and is outside the scope of this Directory Specification.

   NOTE – It is not always possible to detect an invalid superior reference.

d) *Immediate Superior References* – This type of reference is invalid if the referenced DSA does not hold a naming context or a replicated area derived from the naming context with context prefix contained in the reference. Furthermore, usage of this type of reference is only valid when the **operationProgress** component of **ChainingArguments** has the value **notStarted** or **proceeding**. This inconsistency will be detected during the Name Resolution process by inspection of the **operationProgress** and **referenceType** components of **ChainingArguments**.

e) *Supplier References* – This type of reference, which identifies the supplier of a replicated area and optionally the master for the naming context from which the replicated area is derived, is invalid if the referenced DSA is not the shadow supplier for the DSA using the reference (when the **referenceType** component of **ChainingArguments** has the value **supplier**), or if the referenced DSA is not the master for the naming context (when **referenceType** has the value **master**). This inconsistency will be detected during the Name Resolution and operation evaluation phases of operation processing by inspection of the **referenceType** component of **ChainingArguments**.

23.3.2 Reporting of knowledge inconsistencies

If chaining is used in performing a Directory request, all knowledge inconsistencies will be detected by the DSA which holds the invalid knowledge reference, through receiving a **serviceError** with problem **invalidReference**.

If a DSA returns a referral which is based on an invalid knowledge reference, the requester will be returned a **serviceError** with problem **invalidReference** if it uses the referral. How the error condition will be propagated to the DSA which stores the invalid reference is not within the scope of this Directory Specification.

23.3.3 Treatment of inconsistent knowledge references

After a DSA has detected an invalid reference it should try to re-establish knowledge consistency. For example, this can be done by simply deleting an invalid cross reference or by replacing it with a correct one which can be obtained using the **returnCrossRefs** mechanisms.

The way in which a DSA actually handles invalid references is a local matter, and outside the scope of this Directory Specification.

23.4 Knowledge References and contexts

The names in knowledge references shall be the primary distinguished names and may include alternative distinguished values and context information held in **valuesWithContext** for any attribute contributing to any RDN, as described in 9.3 of ITU-T Rec. X.501 | ISO/IEC 9594-2.

Depending on how a knowledge reference is obtained (in particular if a pre-1997 DSA is holding the reference or has been part of the chain through which the reference has been obtained), it is possible that a knowledge reference will not include all possible alternative distinguished names. This may result in a purported name not being recognized as the same name by the holder of the knowledge reference, leading to extra steps in name resolution or, in some situations, inconsistent results or failure of name resolution. The general use of the primary distinguished names, where known, optimizes the ability of the Directory to deal with context variants in names.

24 Hierarchical operational bindings

A hierarchical operational binding is used to represent the relationship between two DSAs holding two naming contexts, one immediately subordinate to the other. In the case of a HOB, the superior DSA holds a subordinate reference to the naming context held by the subordinate DSA; the subordinate DSA holds an immediate superior reference to the naming context held by the superior DSA. The operational binding ensures that the appropriate knowledge information is exchanged and maintained between the two DSAs so that both DSAs are able to behave during the process of Name Resolution and Operation Evaluation as defined in clauses 18 and 19.

24.1 Operational binding type characteristics

24.1.1 Symmetry and roles

The hierarchical operational binding type is an asymmetrical type of operational binding. The two roles in a binding of this type are:

a) the role of the master DSA for the superior naming context, the *superior DSA* (associated with abstract role "A"); and

b) the role of the master DSA for the subordinate naming context, the *subordinate DSA* (associated with abstract role "B").

24.1.2 Agreement

The agreement information exchanged during the establishment of the hierarchical operational binding is a value of **HierarchicalAgreement**. This contains the relative distinguished name of the new context prefix (the **rdn** component) and the distinguished name of the entry immediately superior to the new naming context (the **immediateSuperior** component). This information shall be provided by the DSA that initiates the HOB.

```
HierarchicalAgreement ::= SEQUENCE {
    rdn                  [0]   RelativeDistinguishedName,
    immediateSuperior    [1]   DistinguishedName }
```

The **rdn** shall be the primary RDN, and **immediateSuperior** shall be a primary distinguished name. Context information and all alternative distinguished values shall be included in the **valuesWithContext** component of the **AttributeTypeAndDistinguishedValue** of any RDN, as described in 9.3 of ITU-T Rec. X.501 | ISO/IEC 9594-2.

24.1.3 Initiator

24.1.3.1 Establishment

The establishment of a hierarchical operational binding can be initiated by either role. Initiation by the superior DSA can be caused by an Add Entry operation with the subordinate DSA specified in the **targetSystem** extension, or by administrative intervention. Initiation by the subordinate DSA (which connects a locally existing entry or subtree to the global DIT) is caused by administrative intervention.

24.1.3.2 Modification

The modification of a hierarchical operational binding can be initiated by either role. The superior DSA may issue the modification as a result of a modification of the superior context prefix information. This can be as a result of any of the modification operations, or by administrator intervention.

Either DSA may modify the agreement as a result of a modification of the RDN of the context prefix entry of the subordinate naming context. The superior DSA initiates this modification because of a relative distinguished name being modified higher up the DIT, or because of administrative intervention. The subordinate DSA initiates modification because of a ModifyDN of a context prefix, or because of administrative intervention.

Either DSA may also modify the HOB if the access point information for its naming context changes.

24.1.3.3 Termination

The termination of a hierarchical operational binding can be initiated by either role. Initiation by the superior DSA can be caused by administrative intervention. Initiation by the subordinate DSA can be caused either by a Remove Entry operation that removes the context prefix entry of the subordinate naming context, or by administrative intervention.

24.1.4 Establishment parameters

The establishment parameters for the two roles of a HOB, superior DSA and subordinate DSA, differ. The establishment parameter for the superior DSA role is a value of **SuperiorToSubordinate**, the parameter for the subordinate role, a value of **SubordinateToSuperior**.

24.1.4.1 Superior DSA establishment parameter

The establishment parameter issued by the superior DSA, a value of **SuperiorToSubordinate**, provides the subordinate DSA with information regarding DIT vertices superior to the context prefix of the new naming context (which includes the immediate superior reference) and optionally user and operational attributes for the subordinate context prefix entry and copies of user and operational attributes from the entry immediately superior to the new context prefix.

```
SuperiorToSubordinate ::= SEQUENCE {
        contextPrefixInfo            [0]    DITcontext,
        entryInfo                    [1]    SET OF Attribute OPTIONAL,
        immediateSuperiorInfo        [2]    SET OF Attribute OPTIONAL }
```

The **rdn** in **Vertex** or in **SubentryInfo** shall be the primary RDN, and context information and all other distinguished values shall be included in the **AttributeTypeAndDistinguishedValue** components of the RDN, as described in 9.3 of ITU-T Rec. X.501 | ISO/IEC 9594-2.

24.1.4.1.1 Context prefix information

The **contextPrefixInfo** component of **SuperiorToSubordinate** is a value of type **DITcontext**, this being a sequence of **Vertex** values.

```
DITcontext ::= SEQUENCE OF Vertex

Vertex ::= SEQUENCE {
        rdn             [0]    RelativeDistinguishedName,
        admPointInfo    [1]    SET OF Attribute OPTIONAL,
        subentries      [2]    SET OF SubentryInfo OPTIONAL,
        accessPoints    [3]    MasterAndShadowAccessPoints OPTIONAL }
```

The **contextPrefixInfo** component is the sequence of RDNs that form the distinguished name of the immediate superior of the new context prefix, each RDN (given by the **rdn** component) optionally accompanied by additional information.

The optional **admPointInfo** component of a **Vertex** signals that the DIT vertex is an administrative point and provides, at least, its **administrativeRole** operational attribute.

The subentry information associated with an administrative point is provided by the **subentries** component of a **Vertex**, which is a set of one or more **SubentryInfo** values. Each **SubentryInfo** value is composed of the RDN of the subentry (the **rdn** component) and the attributes of the subentry (the **info** component).

```
SubentryInfo ::= SEQUENCE {
        rdn    [0]    RelativeDistinguishedName,
        info   [1]    SET OF Attribute }
```

The optional **accessPoints** component of a **Vertex** signals that the vertex corresponds to the context prefix of the immediately superior naming context. The superior uses this component to provide the subordinate the information required for its immediate superior reference.

NOTE – The master access point within **accessPoints** is the same as that passed in the **accessPoint** parameter of the Establish and Modify Operational Binding operations.

24.1.4.1.2 Entry information

The optional **entryInfo** component of **SuperiorToSubordinate** is a set of attributes establishing the content of the new context prefix entry.

24.1.4.1.3 Immediate superior entry information

The optional **immediateSuperiorInfo** component of **SuperiorToSubordinate** is a copy of a set of attributes, in particular **objectClass** and **entryACI**, from the entry immediately superior to the new context prefix.

NOTE – This component may be used by the subordinate for optimizing the evaluation of a List request which generates an empty **ListResult** for a base object which is the immediate superior of the subordinate context prefix [see Note of 19.3.1.2.2, item 2)].

24.1.4.2 Subordinate DSA establishment parameter

The establishment parameter issued by the subordinate DSA, a value of **SubordinateToSuperior**, provides the superior DSA with information regarding the subordinate naming context.

```
SubordinateToSuperior ::=   SEQUENCE {
    accessPoints      [0]   MasterAndShadowAccessPoints OPTIONAL,
    alias             [1]   BOOLEAN DEFAULT FALSE,
    entryInfo         [2]   SET OF Attribute OPTIONAL,
    subentries        [3]   SET OF SubentryInfo OPTIONAL }
```

The **accessPoints** component of **SubordinateToSuperior** is used by the subordinate to provide the superior the information required for its subordinate reference.

NOTE 1 – The master access point within **accessPoints** is the same as that passed in the **accessPoint** parameter of the Establish and Modify Operational Binding operations.

The **alias** component of **SubordinateToSuperior** is used to signal to the superior that the subordinate naming context consists of a single alias entry.

The **entryInfo** component of **SubordinateToSuperior** consists of a copy of a set of attributes, in particular **objectClass** and **entryACI**, from the new context prefix entry.

The **subentries** component of **SubordinateToSuperior** is used by the subordinate to pass subentries containing prescriptive ACI to the superior.

NOTE 2 – The latter two components may be used by the superior for optimizing the evaluation of a List or one-level Search request whose base object is the entry immediately superior to the subordinate context prefix.

24.1.5 Modification parameters

For modifications of a HOB, the modification parameter of the superior role, **SuperiorToSubordinateModification**, is **SuperiorToSubordinate**, with the restriction that the **entryInfo** component may not be present; that of the subordinate role is **SubordinateToSuperior**.

```
SuperiorToSubordinateModification  ::=  SuperiorToSubordinate (
    WITH COMPONENTS { ..., entryInfo ABSENT})
```

These parameters are identical (with the restriction noted above) to the corresponding establishment parameters and are used to signal changes occurring to information provided in the establishment parameters subsequent to the establishment of the HOB.

If any component of **SuperiorToSubordinate** (or subsequently **SuperiorToSubordinateModification**) or **SubordinateToSuperior** experiences a change (e.g. the **contextPrefixInfo** component of **SuperiorToSubordinate**), the corresponding component of the modification parameter (e.g. the **contextPrefixInfo** component of **SuperiorToSubordinateModification**) shall be provided in its entirety in the Modify Operational Binding.

24.1.6 Termination parameters

Neither role provides a termination parameter when terminating a HOB.

24.1.7 Type identification

The hierarchical operational binding is identified by the object identifier assigned when defining the **hierarchicalOperationalBinding OPERATIONAL-BINDING** information object in 24.2.

24.2 Operational binding information object Class definition

This subclause defines the hierarchical operational binding type using the **OPERATIONAL-BINDING** information object class template defined in ITU-T Rec. X.501 | ISO/IEC 9594-2.

```
hierarchicalOperationalBinding  OPERATIONAL-BINDING  ::= {
        AGREEMENT                HierarchicalAgreement
        APPLICATION CONTEXTS {
                                 {directorySystemAC} }
        ASYMMETRIC
                ROLE-A  {        -- superior DSA
                        ESTABLISHMENT-INITIATOR         TRUE
                        ESTABLISHMENT-PARAMETER         SuperiorToSubordinate
                        MODIFICATION-INITIATOR          TRUE
                        MODIFICATION-PARAMETER          SuperiorToSubordinateModification
                        TERMINATION-INITIATOR           TRUE }
                ROLE-B  {        -- subordinate DSA
                        ESTABLISHMENT-INITIATOR         TRUE
                        ESTABLISHMENT-PARAMETER         SubordinateToSuperior
                        MODIFICATION-INITIATOR          TRUE
                        MODIFICATION-PARAMETER          SubordinateToSuperior
                        TERMINATION-INITIATOR           TRUE }
        ID                              id-op-binding-hierarchical }
```

24.3 DSA procedures for hierarchical operational binding management

In the following procedures, a new DSE or a mark (i.e. a state indication associated with some item of information) created by a DSA shall be stored in stable storage. By doing so, it is possible for the two DSAs following the procedures below to maintain a consistent understanding of the parameters of the HOB in the presence of communication and end system failures.

In both the **establishment** and **modification** procedure described below, the DSA playing the responding role (i.e. not initiating the establishment or modification) may provide the DSA playing the initiating role with information (e.g. operational attributes) that are not acceptable for one reason or another. The initiating DSA may terminate the operational binding in such cases.

24.3.1 Establishment procedure

24.3.1.1 Establishment initiated by superior DSA

If a DSA evaluates an Add Entry operation with a different DSA specified in the **targetSystem** extension, it shall establish a hierarchical operational binding according to the following procedure. If a DSA, for administrative reasons, wishes to establish a HOB with a subordinate DSA, and it supports the DOP HOB protocol, then the following procedure shall be followed:

1) The superior DSA creates a new DSE of type **subr**, with the name of the new entry, and marks this new DSE as *being added*. The superior DSA generates a unique **bindingID** and stores it with the new DSE.

2) The superior DSA shall send an Establish Operational Binding operation to the subordinate DSA containing the following parameters:

   a) **bindingType** set to **hierarchicalOperationalBindingID**;

   b) **SuperiorToSubordinate** establishment parameter with **contextPrefixInfo** and **entryInfo** components present; all other parameters are optional;

   c) **HierarchicalAgreement** with the **immediateSuperior** component set to the distinguished name of the immediate superior of the new entry and the **rdn** component set to the RDN of the new entry;

   d) the **bindingID**, **myAccessPoint** and **valid** parameters, as appropriate.

3) If the subordinate DSA accepts the operation, it creates the required DSEs of types **glue**, **subentry**, **admPoint**, **rhob** and **immSupr**, as appropriate, to represent the **contextPrefixInfo**; a DSE of type **cp** and **entry** or **alias** to represent the new context prefix object or alias entry; and, as appropriate, a DSE of type

rhob and **entry** to represent the **immediateSuperiorInfo**. It stores the **bindingID** with the DSE of the new context prefix entry and returns a **SubordinateToSuperior** parameter to the superior DSA.

If the subordinate DSA refuses the operation it returns an Operational Binding Error with the appropriate problem value set.

If the naming context already exists and the **bindingID** values for the existing and the new context are the same, the subordinate DSA has already created the requested naming context, in which case the subordinate DSA returns a result to the superior. If the values are not equal, an Operational Binding Error with problem **invalidAgreement** is sent; this means the superior DSA has a permanent knowledge inconsistency that requires correction by an administrator.

4) If the superior DSA receives an error, it deletes the marked DSE of type **subr** and returns an error for the Add Entry operation.

If the superior DSA receives a result, it removes the mark from the DSE that represents the **subr** and returns a result for the Add Entry operation.

If any failure occurs (e.g. communication or end system), the superior DSA shall repeat the steps starting at step 2) until a result or error has been received for each pending establishment of a hierarchical operational binding for which it is the initiator. If the establishment is as a result of an Add Entry operation, and the requester aborts the operation (e.g. by releasing or aborting the application association) before the establishment is complete, the superior DSA shall ignore this event and complete the establishment (which may or may not be successful). In this case, the user will not be informed of the outcome of the Add Entry operation.

NOTE 1 – Marking the subordinate aids recovery and concurrency control. Another user cannot add an entry that is already marked, and the DSA repeats the establish operational binding for all marked subordinates after a failure.

NOTE 2 – With the above procedure, knowledge has only transient inconsistency. It is a local matter how the superior DSA treats unrelated operations that read the subordinate reference while it is marked.

24.3.1.2 Establishment initiated by subordinate DSA

The subordinate DSA may initiate a hierarchical operational binding. This might result from the wish of an administrator to connect a subtree of entries held in the DSA to a certain point in the global DIT. In this case, the subordinate DSA shall establish a HOB according to the following procedure:

1) The subordinate DSA either has a DSE of type **cp** as a part of an existing naming context or it creates a new one. It marks the DSE *being added*, and generates a unique **bindingID** and stores it with the context prefix DSE.

2) The subordinate DSA sends an Establish Operational Binding operation to the superior DSA containing the following parameters:

   a) **bindingType** set to **hierarchicalOperationalBindingID**;

   b) **SubordinateToSuperior** establishment parameter, as appropriate;

   c) **HierarchicalAgreement** with the **immediateSuperior** component set to the distinguished name of the immediate superior of the new entry and the **rdn** component set to the RDN of the new entry;

   d) the **bindingID**, **myAccessPoint** and **valid** parameters, as appropriate.

   If the superior DSA refuses the operation, it returns an Operational Binding Error with the appropriate problem value set.

3) The superior DSA checks that it is master for the immediate superior of the new context prefix entry or returns an Operational Binding Error with problem **roleAssignment**.

4) The superior DSA checks that the requested RDN for the new context prefix is not already in use. If no matching RDN is found using locally held information, but the immediately superior DSE is of type **nssr**, the procedure in 19.1.5 is followed. If no matching RDN is discovered using this procedure, the superior DSA creates a DSE of type **subr**, stores the **bindingID** with it, and returns a result.

If a subordinate reference is found with this RDN, the two values of **bindingID** are compared. If they are equal, a result is returned. The **SuperiorToSubordinate** parameter returned by the superior DSA shall not contain the **entry** component. If the two values of **bindingID** are not equal, an Operational Binding Error with problem **invalidAgreement** is sent; this means the superior DSA has a permanent knowledge inconsistency that requires correction by an administrator.

If a matching RDN is found by exploring an NSSR, an Operational Binding Error with problem **invalidAgreement** is sent; this also means the superior DSA has a permanent knowledge inconsistency that requires correction by an administrator.

5) If the subordinate DSA receives an error, it deletes the new context prefix DSE and its mark. It is a local matter to determine the fate of the entry information from which the context prefix DSE was derived.

   If the subordinate DSA receives a result, it adds the necessary DSEs of types **glue**, **subentry**, **admPoint**, **rhob** and **immSupr**, as appropriate, to represent the **contextPrefixInfo**; and, as appropriate, a DSE of type **rhob** and **entry** to represent the **immediateSuperiorInfo**. The mark of the context prefix DSE is removed.

   If any failure occurs (e.g. communication of end system), the subordinate DSA shall repeat the steps starting at step 2) until a result or error has been received for each pending establishment of a hierarchical operational binding for which it is the initiator.

24.3.2 Modification procedure

The following procedures are defined for modification of a HOB which has been initiated by the procedure detailed in 24.3.1.

24.3.2.1 Modification procedure initiated by superior

This procedure may be invoked as a result of modification operations, as described in 19.1, or as a result of administrative intervention (e.g. to convey changes to the **myAccessPoint**, **agreement** or **valid** parameters of the HOB). Also, if a superior DSA detects changes to the **contextPrefixInfo** or **immediateSuperiorInfo** components of the **SuperiorToSubordinate** value that it supplied to the subordinate DSA, it shall propagate the new information to the subordinate DSA employing the following procedure:

1) Mark the DSE of type **subr** as *being modified*, and if this modification is as a result of a modification to the RDN of the subordinate context prefix entry, a new DSE of type **subr** is added and marked as *being added*.

2) The superior DSA produces a new **bindingID** value from the existing value by incrementing its **version** component. Using this new **bindingID**, it sends a Modify Operational Binding operation to the subordinate DSA with the modification parameter **SuperiorToSubordinateModification**.

3) The subordinate DSA checks the **identifier** component of the **bindingID**. If it has no such agreement with the superior, or if the **version** component is less than the version of the HOB, it shall return an Operational Binding Error with problem **invalidAgreement**.

4) The subordinate DSA may accept the modification to the HOB, modify or rebuild the DSEs representing the context prefix information, update the **version** component of its **bindingID** and return a result. Alternatively, it may return an error and then terminate the agreement.

5) If the superior DSA receives a result, the modification is completed. If this modification is as a result of a modification to the RDN of the subordinate context prefix entry, the new DSE having type **subr** and marked as *being added* has its mark removed, and the old DSE marked as *being modified* is deleted. If not, the mark *being modified* is simply removed.

   If the superior DSA receives an error, the modification has failed. The mark *being modified* is removed. If this modification is as a result of a modification to the RDN of the subordinate context prefix entry, the new DSE having type **subr** and marked as *being added* is removed. If not, the measures taken are outside the scope of this Directory Specification.

   If any failure occurs (e.g. communication or end system), the superior DSA shall repeat the steps starting at step 2) until a result or error has been received for each pending modify of a hierarchical operational binding for which it is the initiator. If the modification is as a result of a ModifyDN operation modifying the RDN of the subordinate context prefix entry, and the requester aborts the operation (e.g. by releasing or aborting the application association) before the modification is complete, the superior DSA shall ignore this event and complete the modification (which may or may not be successful). In this case, the user will not be informed of the outcome of the ModifyDN operation.

24.3.2.2 Modification procedure initiated by subordinate

This procedure may be invoked as a result of administrative intervention (e.g. to convey changes to the **myAccessPoint**, **agreement** or **valid** parameters of the HOB). Also if a subordinate DSA detects changes to the **SubordinateToSuperior** value that it supplied to the superior DSA, it shall propagate the new information to the superior DSA employing the following procedure:

1) Mark the DSE of type **cp** as *being modified*.

2) The subordinate DSA produces a new **bindingID** value from the existing value by incrementing its **version** component. Using this new **bindingID**, it sends a Modify Operational Binding operation to the superior DSA with the modification parameter **SubordinateToSuperior**.

3) The superior DSA checks the **identifier** component of the **bindingID**. If it has no such agreement with the subordinate, or if the **version** component is less than the version of the HOB, it shall return an Operational Binding Error with problem **invalidAgreement**.

4) The superior DSA may accept the modification to the HOB, modify the DSE representing the subordinate reference and return a result. Alternatively, it may return an error and then terminate the agreement.

   In addition, if the superior DSE of the DSE (of type **subr**) to be renamed is of type **nssr,** the DSA shall follow the procedure defined in 19.1.5 (Modify Operations and NSSRs) to ensure that the new name of the entry is unambiguous, before responding to the HOB modification request.

5) If the subordinate DSA receives a result, the modification is completed and it removes the mark. If it receives an error, the measures taken are outside the scope of this Directory Specification.

   If any failure occurs (e.g. communication or end system), the subordinate DSA shall repeat the steps starting at step 2) until a result or error has been received for each pending modify of a hierarchical operational binding for which it is the initiator.

24.3.3 Termination procedure

The following procedures are defined for termination of a HOB which has been initiated by the procedure detailed in 24.3.1.

24.3.3.1 Termination initiated by superior DSA

The termination of a hierarchical operational binding is initiated by the superior DSA only as a result of administrative intervention. The following procedure shall be followed:

1) The superior DSA marks the DSE representing the subordinate reference *being deleted*, so that the subordinate reference is no longer used during Name Resolution.

2) The superior DSA sends a Terminate Operational Binding operation for the hierarchical operational binding to the subordinate DSA. The **version** component of the **bindingID** is omitted by the superior.

3) When the subordinate DSA receives the Terminate Operational Binding, it deletes any information about the hierarchical operational binding and sends a result, unless the **identifier** component of the **bindingID** is unknown, in which case an Operational Binding Error with problem **invalidID**, is returned. It is a local matter to determine the fate of any entry information associated with the subordinate naming context.

4) If the superior DSA receives a result or an Operational Binding Error with problem **invalidID**, it shall delete the DSE marked *being deleted* that represents the subordinate reference associated with the hierarchical operational binding and deletes any information about the operational binding.

   If any failure occurs (e.g. communication of end system), the superior DSA shall repeat the steps starting at step 2) until a result or error has been received for each pending termination of a hierarchical operational binding for which it is the initiator.

24.3.3.2 Termination initiated by subordinate DSA

Termination initiated by the subordinate DSA can be caused by a Remove Entry operation that removes the last entry within the subordinate naming context, the context prefix entry, or as a result of administrative intervention. The following procedure shall be followed:

1) The subordinate DSA marks the context prefix DSE of the naming context *being deleted*.

2) The subordinate DSA sends a Terminate Operational Binding operation for the hierarchical operational binding to the superior DSA. The **version** component of the **bindingID** is omitted by the subordinate.

3) When the superior DSA receives the Terminate Operational Binding, it deletes the DSE that represents the subordinate reference associated with the hierarchical operational binding, deletes any information about the operational binding and sends a result, unless the **identifier** component of the **bindingID** is unknown, in which case an Operational Binding Error with problem **invalidID**, is returned.

4) If the subordinate DSA receives a result or an Operational Binding Error with problem **invalidID**, it shall delete any information about the operational binding.

   NOTE – The fate of the entry information of naming context is a matter local to the subordinate DSA. Since renaming (i.e. moving) a naming context is not allowed by the Modify DN operation, an administrator might, for example, terminate the HOB, select another context prefix for the naming context and reconnect it to another part of the DIT (i.e. establish a new HOB).

   If any failure occurs (e.g. communication of end system), the subordinate DSA shall repeat the steps starting at step 2) until a result or error has been received for each pending termination of a hierarchical operational binding for which it is the initiator.