The Consultative Committee for Space Data Systems

Report Concerning Space Data System Standards

SPACE LINK EXTENSION— APPLICATION PROGRAM INTERFACE FOR TRANSFER SERVICES— SUMMARY OF CONCEPT AND RATIONALE

Informational Report

CCSDS 914.1-G-1

Green Book

January 2006

AUTHORITY

Issue:	Green Book, Issue 1
Date:	January 2006
Location:	Washington, DC

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and reflects the consensus of technical panel experts from CCSDS Member Agencies. The procedure for review and authorization of CCSDS Reports is detailed in the *Procedures Manual for the Consultative Committee for Space Data Systems*.

This document is published and maintained by:

CCSDS Secretariat
Office of Space Communication (Code M-3)
National Aeronautics and Space Administration
Washington, DC 20546, USA

FOREWORD

This document is a technical **Report** for use in developing ground systems for space missions and has been prepared by the **Consultative Committee for Space Data Systems** (CCSDS). The Application Program Interface described herein is intended for missions that are cross-supported between Agencies of the CCSDS.

This **Report** contains background and explanatory material to supplement the CCSDS Recommended Standard and Recommended Practice documents defining the SLE Application Program Interface for Transfer Services (references [10], [11], [13], [14], [15], [16], and [17]).

Through the process of normal evolution, it is expected that expansion, deletion, or modification to this document may occur. This **Report** is therefore subject to CCSDS document management and change control procedures, as defined in the *Procedures Manual for the Consultative Committee for Space Data Systems*. Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be addressed to the CCSDS Secretariat at the address indicated on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- British National Space Centre (BNSC)/United Kingdom.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Federal Space Agency (Roskosmos)/Russian Federation.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Federal Science Policy Office (BFSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- Centro Tecnico Aeroespacial (CTA)/Brazil.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Danish Space Research Institute (DSRI)/Denmark.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Korea Aerospace Research Institute (KARI)/Korea.
- MIKOMTEK: CSIR (CSIR)/Republic of South Africa.
- Ministry of Communications (MOC)/Israel.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic & Atmospheric Administration (NOAA)/USA.
- National Space Program Office (NSPO)/Taipei.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 914.1-G-1	Space Link Extension—Application Program Interface for Transfer Services—Summary of Concept and Rationale, Informational Report, Issue 1	January 2006	Current issue

CONTENTS

CONTENTS (continued)

Section Page

Figure

Table

1 INTRODUCTION

1.1 PURPOSE

This Report presents a summary of supplementary information which underlies the Recommended Standard and Recommended Practice documents for the Application Program Interface (API) for SLE transfer services (references [10], [11], [13], [14], [15], [16], and [17]).

The SLE API provides a high level, communication technology independent interface for exchange of SLE operation invocations and returns between a SLE service user and a SLE service provider. This interface is implemented by software components and is provided to software programs supporting SLE interfaces. Therefore, SLE API Recommended Practice documents make use of software specification techniques and contain detailed definitions of software interfaces.

This Report presents the concepts behind the SLE API and the rationale for the SLE API in a manner that does not require in depth familiarity with software development concepts and techniques. It has been prepared for technical domain experts, who want to understand the general concepts, but might not be interested in all details required for an implementation of the SLE API. Reading this Report might also help to better understand the SLE API Recommended Practice documents. For software developers wishing to implement the SLE API, or to use available implementations of the SLE API, this Report can serve as an initial introduction to the material provided by the SLE API Recommended Practice documents.

1.2 SCOPE

This Report describes the Application Program Interface for SLE Transfer Services. It assumes that the reader is familiar with CCSDS Space Link Extension concepts and has a general understanding of SLE transfer services. Readers not familiar with these topics should read the CCSDS Report *Cross Support Concept – Part 1: Space Link Extension Services* (reference [2]) before proceeding with this report. Knowledge of at least one SLE return service (e.g., the Return All Frames service, reference [5]) and one SLE forward service (e.g., the Forward CLTU Service, reference [8]) would help to understand the more detailed information presented in sections 4 and 5 of this report.

The information contained in this report is not part of the CCSDS Recommended Standard for the SLE Application Program Interface for Transfer Services. In the event of any conflict between the specifications in references [10], [11], [13], [14], [15], [16], and [17] and the material presented herein, the former shall prevail.

1.3 DOCUMENT STRUCTURE

1.3.1 ORGANIZATION OF THIS REPORT

This Report in organized as follows:

a) section 1 presents the purpose and scope of this Report and lists the references used throughout the Report;

b) section 2 provides an overview of the SLE API for transfer services and presents the rationale for the specification of the API;

c) section 3 explains how interoperability between SLE service users and SLE service providers is supported by the SLE API;

d) section 4 summarizes the features provided by the SLE API and discusses the benefits that applications can gain by using the API;

e) section 5 provides a summary of the SLE API object model and outlines how application programs interact with the SLE API;

f) section 6 presents an introduction to the SLE API architecture, identifies API components, and discusses topics related to integration and deployment of SLE API implementations;

g) annex A provides a guide to the SLE API documentation;

h) annex B contains a glossary of important terms used throughout this Report;

i) annex C lists the acronyms used in this Report.

1.3.2 SLE SERVICES DOCUMENTATION

The SLE suite of Recommended Standards is based on the cross support model defined in the SLE Reference Model (reference [3]). The services defined by the reference model constitute one of the three types of Cross Support Services:

a) Part 1: SLE Services;

b) Part 2: Ground Domain Services; and

c) Part 3: Ground Communications Services.

The SLE services are further divided into SLE service management and SLE transfer services.

The basic organization of the SLE services and SLE documentation is shown in figure 1-1. The various documents are described in the following paragraphs.

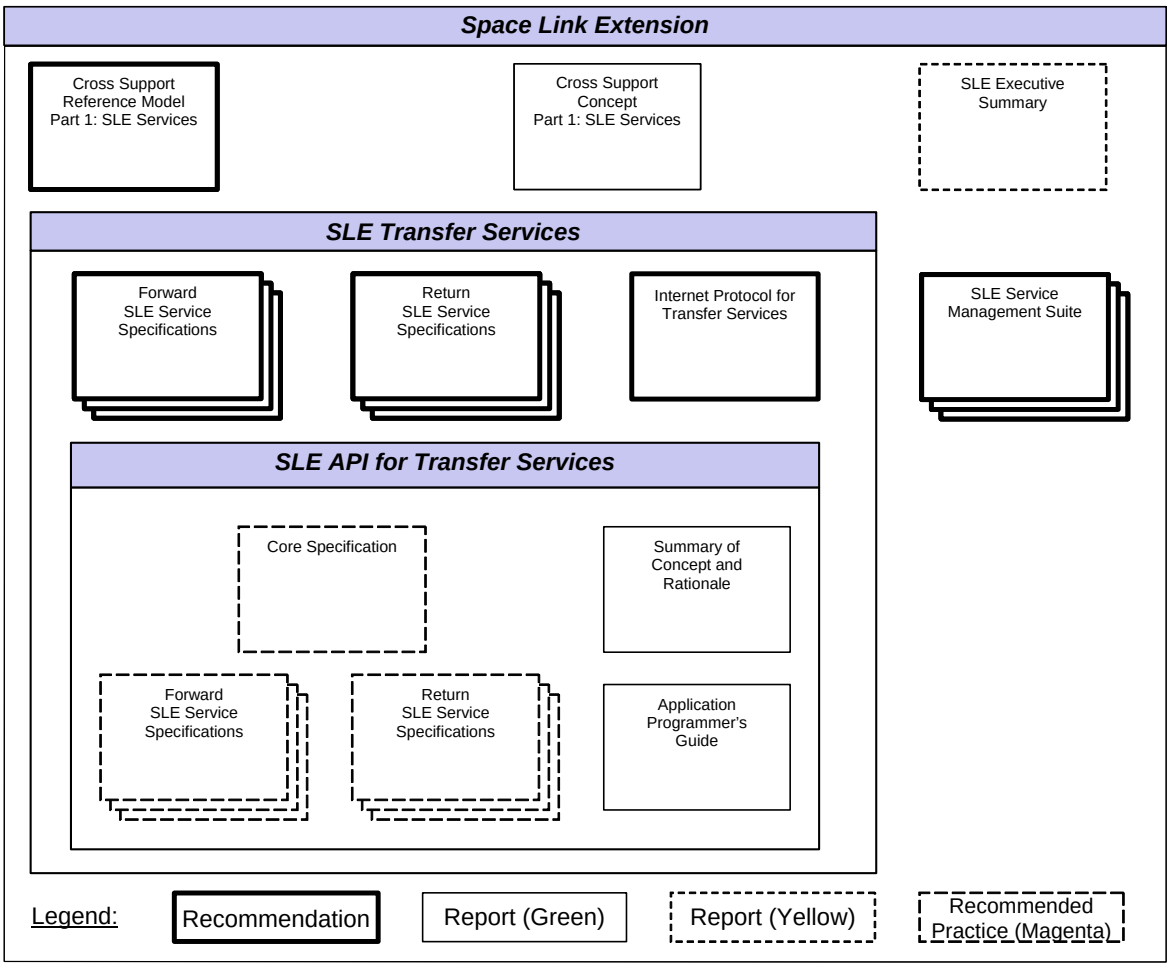


Figure 1-1: SLE Services and SLE API Documentation

a) *Cross Support Reference Model—Part 1: Space Link Extension Services*, a Recommended Standard that defines the framework and terminology for the specification of SLE services.

b) *Cross Support Concept—Part 1: Space Link Extension Services*, a Report introducing the concepts of cross support and the SLE services.

c) *Space Link Extension Services—Executive Summary*, an Administrative Report providing an overview of Space Link Extension (SLE) Services. It is designed to assist readers with their review of existing and future SLE documentation.

d) *Forward SLE Service Specifications*, a set of Recommended Standards that provide specifications of all forward link SLE services.

e) *Return SLE Service Specifications*, a set of Recommended Standards that provide specifications of all return link SLE services.

f) *Internet Protocol for Transfer Services*, a Recommended Standard providing the specification of the wire protocol used for SLE transfer services.

g) *SLE Service Management Specifications*, a set of Recommended Standards that establish the basis of SLE service management.

h) *Application Program Interface for Transfer Services—Core Specification*, a Recommended Practice document specifying the generic part of the API for SLE transfer services.

i) *Application Program Interface for Transfer Services—Summary of Concept and Rationale*, this document.

j) *Application Program Interface for Return Services*, a set of Recommended Practice documents specifying the service-type specific extensions of the API for return link SLE services.

k) *Application Program Interface for Forward Services*, a set of Recommended Practice documents specifying the service-type specific extensions of the API for forward link SLE services.

l) *Application Program Interface for Transfer Services—Application Programmer's Guide*, a Report containing guidance material and software source code examples for software developers using the API.

1.4 REFERENCES

The following documents are referenced in this Report. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this Report are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS documents.

[1] *Procedures Manual for the Consultative Committee for Space Data Systems*. CCSDS A00.0-Y-9. Yellow Book. Issue 9. Washington, D.C.: CCSDS, November 2003.

[2] *Cross Support Concept — Part 1: Space Link Extension Services*. Report Concerning Space Data System Standards, CCSDS 910.3-G-3. Green Book. Issue 3. Washington, D.C.: CCSDS, March 2006.

[3] *Cross Support Reference Model—Part 1: Space Link Extension Services*. Recommendation for Space Data System Standards, CCSDS 910.4-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, October 2005.

[4] *Space Link Extension—Service Management—Service Specification.* Draft Recommendation for Space Data System Standards, CCSDS 910.11-R-1. Red Book. Issue 1. Washington, D.C.: CCSDS, March 2006.

[5] *Space Link Extension—Return All Frames Service Specification*. Recommendation for Space Data System Standards, CCSDS 911.1-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, December 2004.

[6] *Space Link Extension—Return Channel Frames Service Specification*. Recommendation for Space Data System Standards, CCSDS 911.2-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, December 2004.

[7] *Space Link Extension—Return Operational Control Fields Service Specification*. Recommendation for Space Data System Standards, CCSDS 911.5-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, December 2004.

[8] *Space Link Extension—Forward CLTU Service Specification*. Recommendation for Space Data System Standards, CCSDS 912.1-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, December 2004.

[9] *Space Link Extension—Forward Space Packet Service Specification*. Recommendation for Space Data System Standards, CCSDS 912.3-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, December 2004.

[10] *Space Link Extension—Internet Protocol for Transfer Services*. Draft Recommendation for Space Data System Standards, CCSDS 913.1-R-1. Red Book. Issue 1. Washington, D.C.: CCSDS, October 2005.

[11] *Space Link Extension—Application Program Interface for Transfer Services—Core Specification*. Draft Specification Concerning Space Data System Standards, CCSDS 914.0-M-0.1. Draft Magenta Book. Issue 0.1. Washington, D.C.: CCSDS, October 2005.

[12] Space Link Extension—Application Program Interface for Transfer Services—Application Programmer's Guide. Report Concerning Space Data System Standards, CCSDS 914.2-G-1. Green Book. Issue 1. Washington, D.C.: CCSDS, January 2006.

[13] *Space Link Extension—Application Program Interface for Return All Frames Service*. Draft Specification Concerning Space Data System Standards, CCSDS 915.1-M-0.1. Draft Magenta Book. Issue 0.1. Washington, D.C.: CCSDS, October 2005.

[14] *Space Link Extension—Application Program Interface for Return Channel Frames Service*. Draft Specification Concerning Space Data System Standards, CCSDS 915.2-M-0.1. Draft Magenta Book. Issue 0.1. Washington, D.C.: CCSDS, October 2005.

[15] *Space Link Extension—Application Program Interface for Return Operational Control Fields Service*. Draft Specification Concerning Space Data System Standards, CCSDS 915.5-M-0.1. Draft Magenta Book. Issue 0.1. Washington, D.C.: CCSDS, October 2005.

[16] *Space Link Extension—Application Program Interface for the Forward CLTU Service*. Draft Specification Concerning Space Data System Standards, CCSDS 916.1-M-0.1. Draft Magenta Book. Issue 0.1. Washington, D.C.: CCSDS, October 2005.

[17] *Space Link Extension—Application Program Interface for the Forward Space Packet Service*. Draft Specification Concerning Space Data System Standards, CCSDS 916.3-M-0.1. Draft Magenta Book. Issue 0.1. Washington, D.C.: CCSDS, October 2005.

[18] *Programming Languages—C++*. International Standard, ISO/IEC 14882:2003. 2nd ed. Geneva: ISO, 2003.

[19] *The COM/DCOM Reference*. COM/DCOM Product Documentation, AX-01. San Francisco: The Open Group, 1999. <http://www.opengroup.org/products/publications/catalog/ax01.htm>

[20] Clemens Szyperski, with Dominik Gruntz and Stephan Murer. *Component Software: Beyond Object-Oriented Programming*. 2nd ed. Component Software Series. Reading, Massachusetts/New York, New York: Addison-Wesley/ACM Press, 2002.

[21] *Unified Modeling Language (UML)*. Version 1.5, formal/2003-03-01. Needham, Massachusetts: Object Management Group, March 2003. <http://www.omg.org/technology/documents/modeling_spec_catalog.htm>

[22] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. 2nd ed. The Addison-Wesley Object Technology Series. Reading, Massachusetts: Addison-Wesley, 2004.

[23] Grady Booch, Ivar Jacobson, and James Rumbaugh. *The Unified Modeling Language User Guide*. Reading, Massachusetts: Addison-Wesley, 1999.

[24] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 2nd ed. Reading, Massachusetts: Addison-Wesley, 1999.

[25] Don Box. *Essential COM*. The DevelopMentor Series. Reading, Massachusetts: Addison-Wesley, 1997.

[26] *Information Technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. International Standard, ISO/IEC 7498-1:1994. 2nd ed. Geneva: ISO, 1994.

[27] *Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*. International Standard, ISO/IEC 8824-1:2002. 3rd ed. Geneva: ISO, 2002.

[28] *Information Technology—ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*. International Standard, ISO/IEC 8825-1:2002. 3rd ed. Geneva: ISO, 2002.

[29] *Information Technology—Open Systems Interconnection—The Directory: Public-Key and Attribute Certificate Frameworks*. International Standard, ISO/IEC 9594-8:2001. 4th ed. Geneva: ISO, 2001.

[30] *Secure Hash Standard*. Federal Information Processing Standards Publication 180-1. Gaithersburg, Maryland: NIST, 1995.

[31] J. Postel. Transmission Control Protocol. STD 7. Reston, Virginia: ISOC, September 1981.

[32] J. Postel. *Internet Protocol*. STD 5. Reston, Virginia: ISOC, September 1981.

[33] R. Braden, ed. *Requirements for Internet Hosts*. STD 3. Reston, Virginia: ISOC, October 1989.

2 OVERVIEW

2.1 INTRODUCTION

The SLE Reference Model (reference [3]) and the Recommended Standards for SLE transfer services (references [5], [6], [7], [8], [9]) define an abstract model for SLE service production and provision, which is illustrated in figure 2-1.

NOTE – The following description of the model is informal and in outline only. Formal and complete specifications, based on the *Abstract Service Definition Conventions* (reference [26]) can be found in the Recommended Standards referenced above.

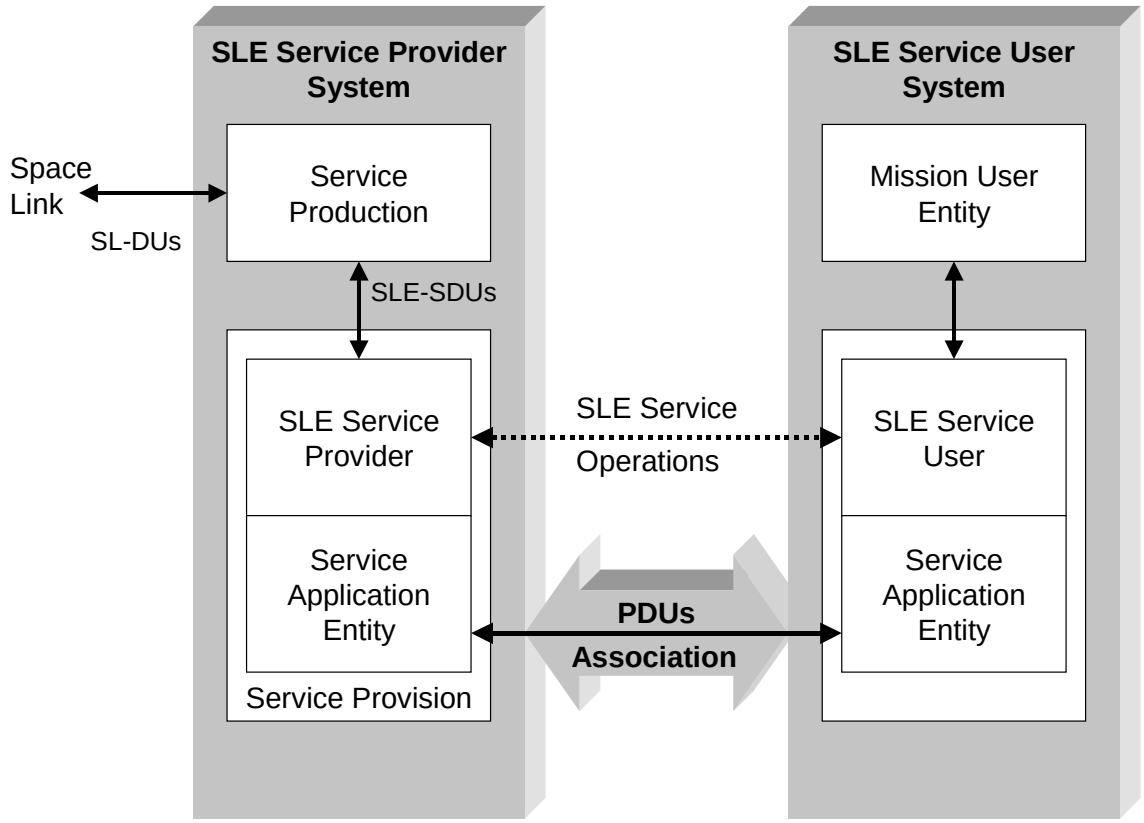


Figure 2-1: SLE Service Production and Provision

SLE services are provided by a SLE service provider in a SLE Complex to a Mission User Entity (MUE) in a Mission Data Operation System (MDOS) or to a service user in a different SLE Complex.

On the provider side, SLE Recommended Standards distinguish between service **production** and service **provision**.

Return service production is concerned with acquisition of Space Link Data Units (SL-DU) from the space link and their annotation to form a stream of SLE-Service Data Units (SLE-SDU) independent of any particular instance of service. In contrast, **return service provision** is concerned with delivering the stream of SLE-SDUs to a service user.

Forward service production is concerned with extraction of SL-DUs from one or more streams of SLE-SDUs received from the service user and their multiplexing onto the space link according to the applicable space link protocol, annotation in the SLE-SDU, and configuration set up by service management. **Forward service provision** is concerned with receiving one stream of SLE-SDUs from the service user.

Service provision addresses such matters as when service is provided (e.g., service start and stop times) and how service is provided (e.g., which events are notified to the user).

The service interface between the user and the provider is specified in terms of SLE operations defined in the Recommended Standards for SLE transfer services. These operations are realized by mapping them to protocol data units (PDU) that can be exchanged by means of the underlying communications service. The Recommended Standards for SLE transfer services conceptualize such mapping in two parts. First, SLE service operations are mapped to SLE-PDUs, which generally correspond to the invocation or return of an SLE operation. Second, SLE-PDUs are mapped to protocol data units that can be exchanged by means of the underlying communications service. The mapping of SLE-PDUs to an underlying communications service is intentionally outside the scope of these Recommended Standards (e.g., so that SLE transfer services may be mapped to more than one communications technology).

As illustrated in figure 2-1, from the point of view of the service provider or service user, the interaction between the user and provider is in terms of operations, but from the point of view of the application entities that implement the SLE protocol, what is exchanged are protocol data units. In any real system, these protocol data units must be formatted according to the requirements of the underlying data communication service and are exchanged across a technology specific association between the service user and the service provider.

The prime objective of the **SLE Application Interface for Transfer Services** (SLE API) is to enable development of reusable software packages, which provide a high level, communication technology independent interface to SLE application programs. The SLE API offers services for exchange of SLE operation invocations and returns between a SLE service user and a SLE service provider. It specifies interfaces supporting SLE service user applications and SLE service provider applications. Once developed, SLE API software packages can be supplied to organizations wishing to introduce SLE services for integration with their user or production facilities, thus minimizing their investment to buy into SLE support.

In order to meet this objective, the core specification of the SLE API (reference [10]) defines:

a) a ‘technology abstraction layer’, which encapsulates all data communication interfaces;

b) an additional support layer, which covers all those aspects of SLE transfer service provisioning that can be clearly separated from service production and service use; and

c) an Application Program Interface, specified in the C++ programming language, which enables SLE applications to use the services of SLE API packages.

The relationship between the layers defined in the SLE API Recommended Standard and the model for SLE service production and provision is depicted in figure 2-2.

NOTE – Software packages implementing the SLE API must be integrated into application programs supporting SLE interfaces. A SLE provider application and a SLE user application as shown in figure 2-2 will generally comprise many other elements, including other software programs and hardware devices. As the SLE API Recommended Standard makes no assumptions concerning the scope of the application program using the SLE API, such elements are not shown in the figure.

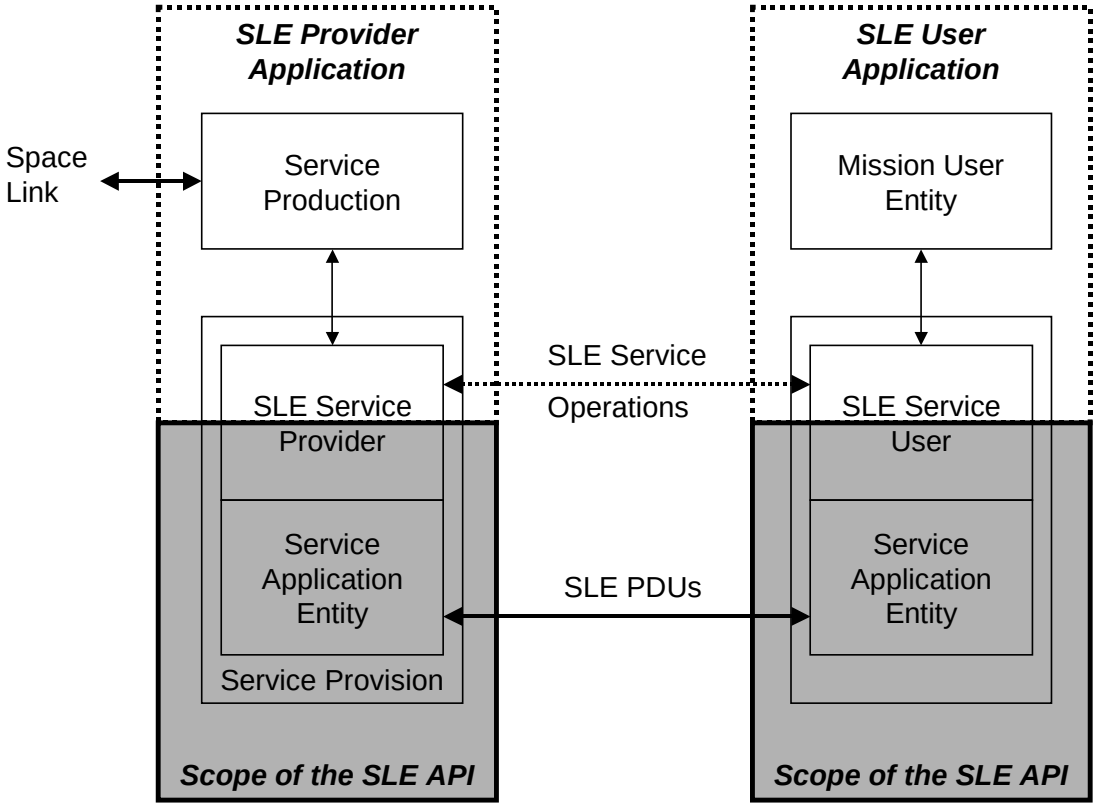


Figure 2-2: Scope of the SLE API

2.2 SLE API LAYERS

The SLE API for transfer services consists of two distinct layers, the API Proxy and the API Service Element, as shown in figure 2-3.

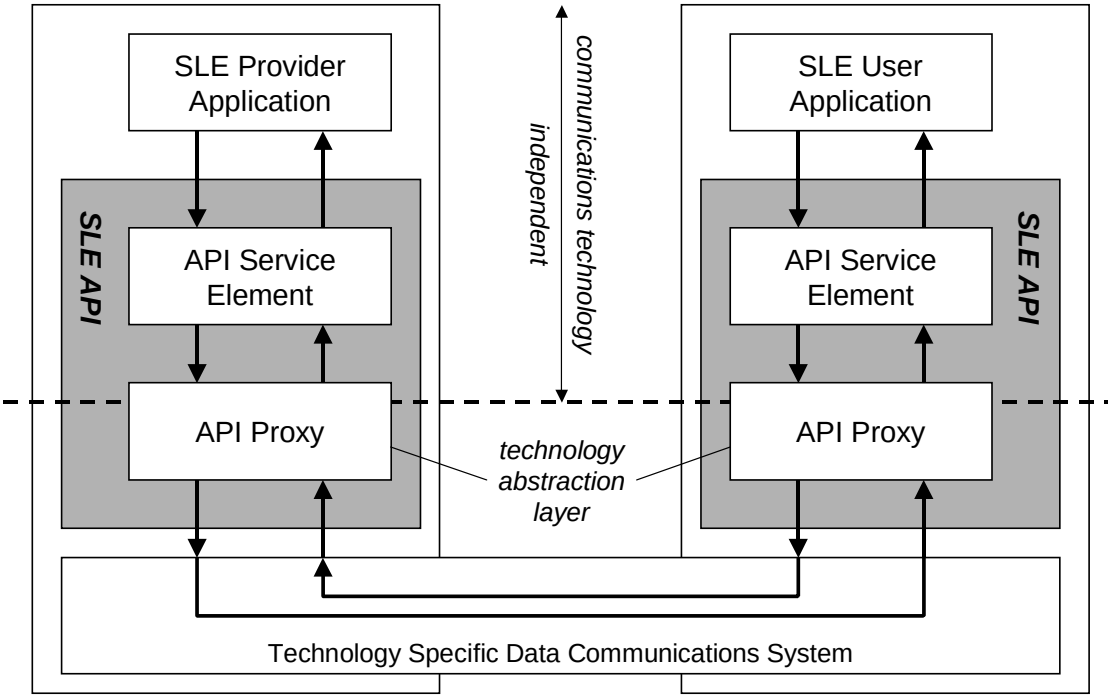


Figure 2-3: Layers of the SLE API

The API Proxy represents the technology abstraction layer. It provides a, technology independent interface to the data communications system used for exchange of SLE protocol data units. It makes higher layers of the API and SLE applications independent of the specific communications technology and shields applications from technology changes.

For the purpose of the SLE API specification, the API Proxy is understood to include all data communications software and all tools required for syntax translation. The associated features might be implemented by software products used by the proxy or by the proxy itself. The SLE API Recommended Standard does not constrain the technology used by a proxy for data communication and syntax translation in any way. However, it expects that a conforming API Proxy provide the quality of service required by the Recommended Standards for SLE transfer services, i.e., that protocol data units be transferred:

a) in sequence;

b) completely and with integrity;

c) without duplication;

d) with flow control; and

e) with notification to the application in the event that communications between the service user and the service provider is disrupted, possibly resulting in loss of data.

If the technology used by the proxy does not provide these characteristics, it is expected that the proxy implement suitable procedures to achieve the desired behavior. An API Proxy can use a connection-less communications technology. However, the API Recommended Standard requires that the API Proxy support the notion of an association between a SLE service user and a SLE service provider as defined by the Recommended Standards for SLE transfer services, including the operations for binding (establishment of an association) and unbinding (release of an association).

The API Service Element implements those aspects of SLE transfer service provisioning that can be clearly separated from service production and service use, offloading applications from lower level aspects of the SLE transfer service protocol. It enforces conformance to the state tables defined by the Recommended Standards for SLE transfer services and checks parameters of SLE operations on completeness, consistency, and range. These features support early detection of any potential problems in SLE applications and protect the application software from errors induced by a peer system.

Because the API Service Element uses the services of the API Proxy it is itself technology independent. Independence from a specific communications technology can be achieved by using the API Proxy alone. The API Recommended Standard does not exclude that applications use the API Proxy directly. The application would have to implement the functionality allocated to the API Service Element and would have to provide the interfaces, which the API Proxy requires. An example of an application that would use the API Proxy directly rather than via the API Service Element is the bridge module in a SLE gateway discussed in 3.4.

2.3 RATIONALE FOR THE SLE API

As mentioned earlier, the prime objective of the SLE API Recommended Standard is to enable development of reusable software packages, which provide a high level, communication technology independent interface to SLE application programs for exchange of SLE operation invocations and returns between a SLE service user and a SLE service provider.

Encapsulation of communication technology specific interfaces within the proxy component, enables interoperability between a service user and a service user by use of a common proxy implementation, without the need to agree on a single technology for SLE services. In addition, the SLE API protects applications from changes in the communications technology.

NOTE – Approaches to achieve interoperability via the SLE API are discussed to more detail in section 3 of this Report.

Use of readily made and tested packages implementing the SLE API is expected to greatly simplify development of application programs supporting SLE interfaces. In addition, use of

tested implementations can considerably reduce the time needed for testing, and the probability of interoperability problems. In order to achieve a maximum level of support for applications, the SLE API extends beyond handling of data communication interfaces offloading applications from most of the lower levels of the aspects of the SLE transfer service protocol.

NOTE – A summary of the features implemented by the SLE API and of the services it provides to applications can be found in section 4 of this Report.

In order to simplify use of SLE API implementations, the SLE API Recommended Practice documents define an object model, which directly mirrors the concepts described in the SLE Reference Model (reference [3]) and the Recommended Standards for SLE transfer services (references [5], [6], [7], [8], [9]). Application programmers, who are familiar with the SLE service concepts, will therefore find the API intuitive and easy to use.

NOTE – A high level description of the SLE API object model is provided in section 5 of this Report.

In order to simplify integration and deployment of SLE implementations the architecture of the SLE API follows a component based design approach. This approach enables delivery and integration of binary API components instead of source code and allows integration of API components from different sources. The latter feature is particularly important for the concept of achieving interoperability by exchange of API Proxy components. At the same time, the architecture of the SLE API minimizes the dependency on any specific platform or runtime environment.

NOTE – An introduction to the design techniques used for the SLE API and an overview of the API architecture is provided in section 6 of this Report.

2.4 TECHNOLOGY MAPPING

In addition to the technology independent SLE API Recommended Practice documents (references [11], [13], [14], [15], [16], and [17]), the Recommended Standard on ‘Internet SLE Protocol’ (reference [10]) specifies the mapping of the SLE API to CCSDS recognized technologies.

At the time of publication, CCSDS had defined a single technology mapping, which uses TCP/IP (see references [31], [32], and [33]) as data communication service and the Abstract Syntax Notation One (references [27] and [28]) for data encoding. The Recommended Standard includes a very simple application layer protocol referred to as the ‘Internet SLE Protocol One’ (ISP1). The choice of the name explicitly acknowledges that use of this protocol is just one of many options supported by the SLE API. Nevertheless, specification of an ‘inter proxy protocol’ based on a widely supported industry standard, further promotes development of products for the SLE API and for SLE applications.

2.5 IMPLEMENTATION STATUS

An initial version of the SLE API specification was developed in early 1999 and subsequently implemented by two independent teams. Both implementations support version 1 of the SLE transfer services Return All Frames (RAF), Return Channel Frames (RCF), and Forward CLTU and implement the technology mapping to TCP/IP and ASN.1 as specified in reference [10]. Extensive interoperability tests were performed between the two implementations on the level of the API implementations alone and between SLE service user and SLE service provider systems integrating the SLE API implementations.

Following the initial development, API implementations were ported to further platforms and integrated into further applications. At the time of writing, API implementations existed on Sun/Solaris, OpenVMS, and Windows NT. These implementations have been successfully integrated into a variety of applications, including mission control systems, ground station equipment for telemetry and telecommand processing, and spacecraft simulators.

During an additional development stage, one of the implementations was extended to implement the Return Operational Control Fields (ROCF) and Forward Space Packet (FSP) services as well as version 2 of the RAF, RCF and CLTU services.

During the development and testing of the initial implementations, the API specification was continuously updated to correct errors identified and to apply improvements where necessary. The specification provided in references [10], [11], [13], [14], [15], [16], and [17] essentially reflects the final status of the initial specification, for which tested implementations are available. Only a limited number of modifications were applied to adopt changes to the Recommended Standards for SLE services and to apply improvements due to lessons learned during the initial development and porting of the API implementations.

3 INTEROPERABILITY CONCEPTS

3.1 INTRODUCTION

The SLE API supports three different approaches to achieve interoperability between a SLE service user and a SLE service provider:

a) use of a common API Proxy implementation;

b) use of an agreed data communications technology and an agreed mapping of the SLE transfer services to that technology; and

c) use of a gateway.

Each of these concepts is briefly explained in the following subsections.

3.2 COMMON API PROXY IMPLEMENTATION

The architecture of the SLE API allows achieving interoperability by use of a common API Proxy implementation, as shown in figure 3-1. The figure assumes that organizations providing SLE services deliver an implementation of the API Proxy to their customers, which supports the communications technology used by the provider organization. The SLE API allows a SLE user application to integrate different proxies such that it could use SLE services from several providers concurrently.

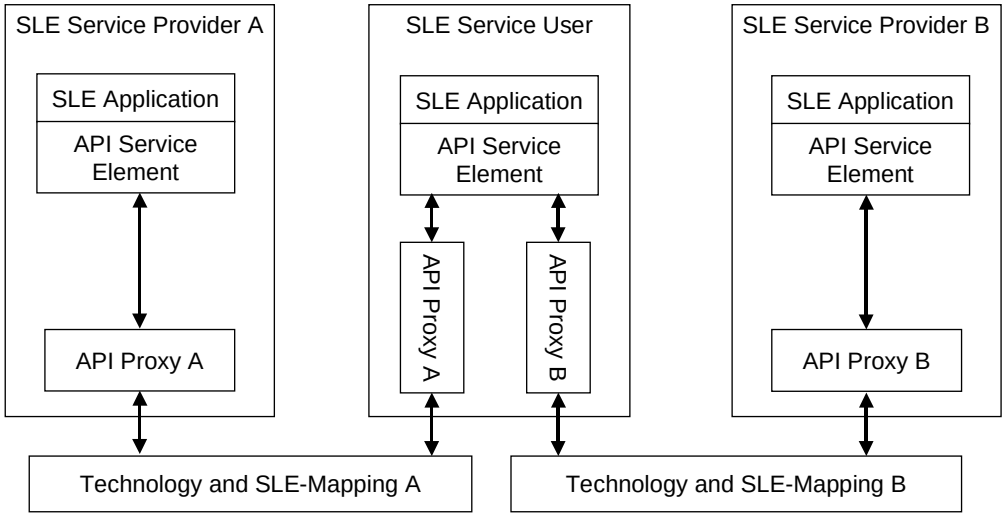


Figure 3-1: Interoperability Using a Common Proxy Implementation

Obviously, this approach requires that the communications infrastructure needed by the proxies be available on the system using the proxy. It is noted that there are variations of this approach. A SLE service user could also provide the proxy to a SLE service provider. In addition, an organization providing SLE services might distribute the complete SLE API to its customers.

3.3 AGREED TECHNOLOGY AND MAPPING

If a SLE service user and a SLE service provider agree on use of a specific communications technology and on the mapping of SLE service operations to that technology, interoperability can be achieved 'on the wire', as illustrated in figure 3-2.

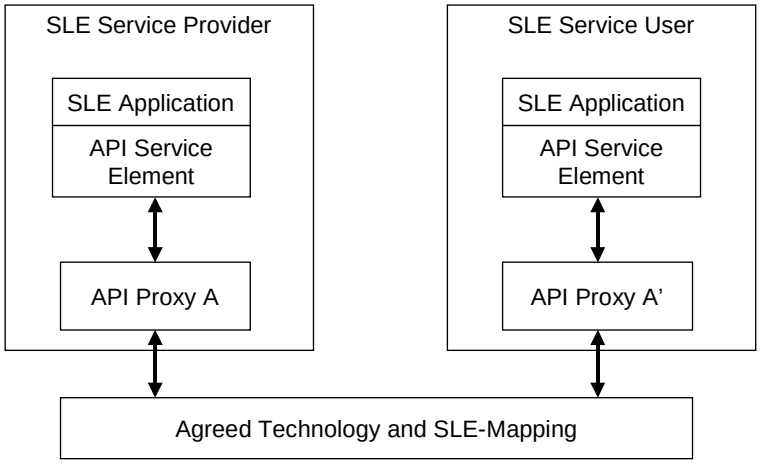


Figure 3-2: Interoperability Using an Agreed Technology and Mapping

In such a configuration, the proxies A and A' must be implemented to the same specification, e.g., the CCSDS Recommended Standard for technology mapping (reference [10]), but need not be of the same make.

3.4 GATEWAY

Finally, interoperability can be achieved by means of an application-layer gateway as sketched in figure 3-3. The API Recommended Standard does not include specification of the gateway, but the design of the SLE API has been performed with the objective to make building of a gateway possible and all perceived requirements of a gateway have been taken into account.

A gateway for SLE transfer services should be located at the lowest possible technology-independent layer. Therefore, it is anticipated that a gateway is built from API Proxy components supporting the technologies, between which the gateway shall mediate. In addition, a simple bridge module will be required to initialize the proxies and to pass SLE operation invocations and returns between them. The bridge module would have to implement the interfaces required by the API Proxy.

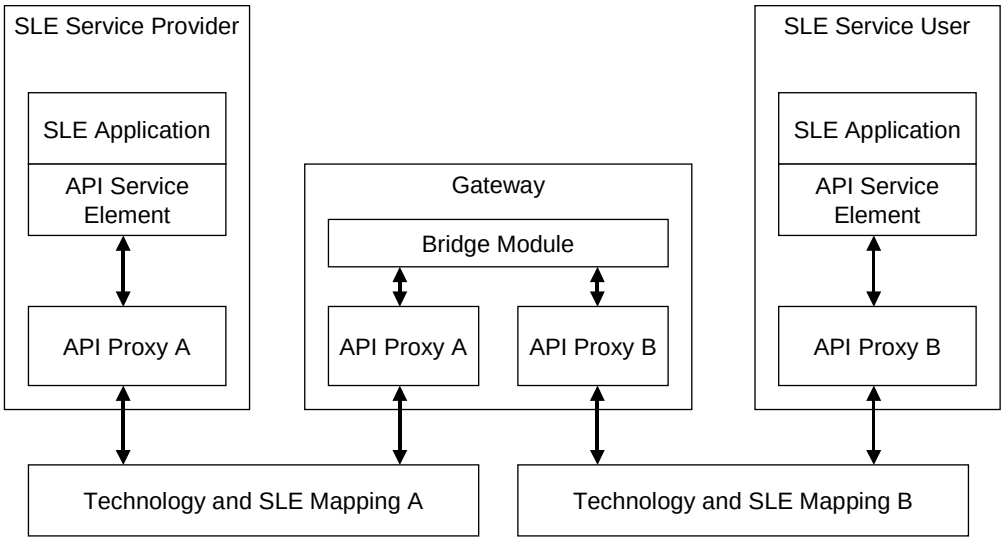


Figure 3-3: Interoperability Using a Gateway

To enable building of a gateway, the design of the interface between the API Proxy and the API Service Element ensures that the API Proxy remains independent from service instance management. This feature allows operation of a gateway, which does not need to know the service instances it supports in advance.

4 FEATURES PROVIDED BY THE SLE API

4.1 OVERVIEW

An important objective of the SLE API is to simplify implementation of systems providing or using SLE transfer services by enabling development of software packages that can be re-used by various application programs. Features that are provided by software packages implementing the SLE API include the following:

a) The API Proxy fully encapsulates handling of data communication interfaces, including association establishment and release, conversion of SLE operations into PDUs, and translation of data structures between the local syntax and the representation on the network.

b) The API implements access control and authentication in a manner that conforms to the requirements stipulated by the SLE Recommended Standards for transfer services.

c) The API implements the state tables defined by the Recommended Standards for SLE transfer services, checks all invocation and return parameters with respect to range, completeness, and consistency, and offloads the application from processing of the invocation identifier required by the Recommended Standards for SLE transfer services.

d) The API provides specific support to SLE service provider applications including management of service instances, status reporting, and processing of the GET-PARAMETER operation.

e) For return link services, the API fully handles the transfer buffer, such that applications only need to deal with TRANSFER-DATA operations. Processing by the API includes controlled loss of data in the timely online delivery mode and flow control in the online complete and offline delivery modes.

f) For forward link services, the API provides flow control for TRANSFER-DATA invocations. On the provider side, it includes specific support to track progress of service production and generate notifications to the SLE service user.

g) The API supports interfaces to log important events and to notify the application of alarms.

h) As an implementation option, the API Recommended Practice documents define interfaces to generate diagnostic traces for testing or for investigation of communications problems.

The following subsections provide a brief description of these features and discuss the benefits that they provide to applications.

4.2 HANDLING OF DATA COMMUNICATION INTERFACES

Data communication interfaces are fully encapsulated by the API proxy layer. As explained earlier (see 2.2 and section 3), the prime objective of this approach is to enable interoperability without the need to agree on use of a single data communication technology. At the same time, the API offloads application programs from having to deal with the details of data communication interfaces.

The services provided by the API cover all aspects related to data communications, including:

a) mapping of logical port identifiers to technology specific address information;

b) establishment and release of associations between the service user and the service provider;

c) monitoring of the state of health of the data communications link and notification of link failures to the application;

d) conversion of SLE operation invocations and returns into protocol data units according to the requirements of the specific technology used, including translation between the local data representation and the representation used on the network;

e) transmission of protocol data units in sequence, without omission or duplication, and with flow control;

f) implementation of suitable procedures supporting the SLE PEER-ABORT operation.

4.3 SECURITY

4.3.1 GENERAL

The SLE API provides support for:

a) access control at system level and to service instances; and

b) authentication of the peer identity.

Other security features such as privacy or integrity must be supported by the data communication technology used by the proxy, if they are required.

4.3.2 ACCESS CONTROL

The SLE API provides access control on system level by means of an access control list, which must be defined in the configuration database of the API Proxy. This list contains the identifiers of all SLE applications that are allowed to communicate with the application hosting the API Proxy. The API Proxy establishes associations only with applications defined in that list. In addition, the API Service Element verifies that a service instance, to which a SLE service user requests access, has actually been scheduled for that user. If the service instance is not scheduled for the user, the API rejects the BIND invocation. All access violations generate an alarm notified to the application.

4.3.3 AUTHENTICATION

Authentication of the peer identity can be supported by the communication technology used by the API Proxy. For cases, in which authentication of the peer identity is not supported by the data communication technology, the API provides basic authentication according to the 'Protected Simple Authentication' procedure (Protected 1) specified by reference [29]. For generation and verification of credentials, the API Recommended Standard specifies use of the Secure Hash Function (SHF-1) defined by reference [30].

These services are fully provided by the API Proxy in manner that is transparent to the application software. The proxy supports definition of a password and of an authentication mode for every peer application entered into the access control list. The authentication mode can have the following values:

a) do not apply authentication;

b) apply authentication for the BIND invocation PDU and the BIND return PDU;

c) apply authentication for all PDUs except for PEER-ABORT.

4.4 PROCESSING OF PROTOCOL DATA UNITS

The API implements the state tables defined by the Recommended Standards for SLE transfer services and verifies that operation invocations and returns received from a peer application are valid in the current state of the service instance. In case of protocol errors, the API generates a PEER-ABORT invocation, transmits it to the peer, and terminates the association in an abortive manner. In addition, the API informs the application, that the current association was aborted.

If a PDU is valid according to the state table, the API verifies that the invocation or return parameters are complete, consistent and in the range specified by the applicable Recommended Standard. Incorrect invocation parameters for confirmed operations are rejected autonomously using the associated return PDU and the appropriate diagnostic code. If the parameters of unconfirmed invocations or of operation returns are incorrect, the API aborts the association using the PEER-ABORT operation.

NOTE – The API can only verify parameters against the limits defined in the Recommended Standards for SLE transfer services. Further constraints imposed by service management must be checked by the application. In addition, the API cannot perform checks that require knowledge of the service production. Such verifications must also be implemented in the application software. The API Recommended Practice documents provide a detailed list of the tests performed by the API for every operation.

The SLE API performs the same checks for invocations and returns issued by the local application. If an operation or return issued by the application violates the state table or contains incorrect parameters, the API rejects the request but does not abort the association.

These features support early detection of any potential problems in SLE applications and protect the application software from errors induced by a peer system.

In addition to verification of PDUs the API offloads the application from handling of the invocation identifier required by SLE Recommended Standards for transfer services. The API:

a) ensures that all invocation PDUs transmitted have a unique invocation identifier;

b) ensures that the invocation identifier in transmitted return PDUs matches the invocation identifier in the corresponding invocation PDU;

c) checks that the invocation identifier in incoming invocation PDUs is unique;

d) associates incoming return PDUs with transmitted invocations using the invocation identifier; and

e) reacts to errors according to the specifications in the Recommended Standards for SLE transfer services.

4.5 SPECIFIC SUPPORT FOR PROVIDER APPLICATIONS

4.5.1 SERVICE INSTANCE MANAGEMENT

The SLE API provides specific support to provider applications for management of service instances. The application can create and configure one or more service instance objects for different service types. Among other things, configuration parameters for a service instance include the scheduled provision period, an identifier of the port at which the service is provided, and the identification of the user for which the service instance is made available.

Once the service instance has been configured, the API processes BIND invocations for the service instance, which it receives at the specified port. The API verifies that:

a) the time at which the BIND invocation arrives is within the scheduled provision period;

b) the identification of the service user in the BIND invocation matches the one specified by the application; and

c) the service instance is not already bound.

NOTE – If use of authentication is specified for the service user, the API also performs authentication as described in 4.3.3.

Only BIND invocations that pass all checks are forwarded to the application. If the checks performed by the API fail, the API autonomously rejects the request using a BIND return with a negative result and the appropriate diagnostic.

As specified by the Recommended Standards for SLE transfer services, the API allows users to un-bind and re-bind to a service instance during the scheduled provision period. At the

end of the scheduled provision period, the API terminates processing of the service instance, aborting the association if it is still active, and informs the application.

NOTE – Further information on the concept of service instance objects in the SLE API is provided in 5.2.

4.5.2 STATUS REPORTING

The SLE API autonomously processes the SLE operations SCHEDULE-STATUS-REPORT and STATUS-REPORT defined by the Recommended Standards for SLE transfer services.

As far as possible, the API derives the parameters needed for the status report from the operation invocations and returns exchanged between the service user and service provider applications. This applies in particular to all statistical information.

Status report parameters that are related to service production must be provided by the application. For this purpose, the API provides specific interfaces by which such parameters can be updated whenever their value changes.

NOTES

1 Examples of RAF status parameters that are derived by the API include `number-of-error-free-frames-delivered` and `number-of-frames-delivered`.

2 Examples of RAF status parameters that must be updated by the application include `frame-sync-lock-status` and `production-status`.

4.5.3 PROCESSING OF THE OPERATION GET-PARAMETER

The SLE API processes the operation GET-PARAMETER without involvement of the application. As for status reporting, service parameters that can be retrieved by the GET-PARAMETER operation are either derived from other operation invocations and returns exchanged between the user and the provider or must be supplied by the application as a configuration parameter.

4.6 SPECIFIC SUPPORT FOR RETURN LINK SERVICES

The SLE API implements all processing required to support the transfer buffer specified by the Recommended Standards for SLE return link services. Provider applications pass TRANSFER-DATA and SYNC-NOTIFY invocations to the API. The API inserts these invocations into a buffer and transmits the complete buffer when the configured transfer buffer size is reached. On the user side, the API disassembles the buffer and passes the invocations to the user application in the sequence the data units were passed by the provider.

In the timely online delivery mode, the API additionally handles the 'latency limit' and implements the procedure for controlled loss of data specified by the Recommended Standards for SLE return link services. I.e.:

a) if the provider application stops supplying TRANSFER-DATA and SYNC-NOTIFY invocations, the API transmits the transfer buffer latest after the timeout defined by the 'latency limit';

b) when a transfer buffer is due for transmission, the API inserts it into a send queue and starts a new buffer. When the new buffer is due for transmission and the previous buffer could not yet be transmitted because the communication link is congested, the API discards the buffer waiting for transmission, inserts a notification 'data discarded due to excessive backlog' at the beginning of the new buffer, and queues the new buffer for transmission.

In the complete online and in the offline delivery modes, the API also uses the transmit buffer for data buffering, but requests the application to suspend delivery of TRANSFER-DATA invocations if a buffer cannot be transmitted because of limited bandwidth. As soon as the queued buffer has been transmitted, the API notifies the application, that delivery of TRANSFER-DATA invocations can be resumed.

NOTE – The Recommended Standards for the RAF service (reference [5]), the RCF service (reference [6]) and ROCF service (reference [7]) additionally define an ‘online frame buffer’ used in the complete online delivery mode. This buffer is not supported by the API and must be implemented by the application.

4.7 SPECIFIC SUPPORT FOR FORWARD LINK SERVICES

For forward link services the user side API provides a flow control mechanism, which ensures that TRANSFER-DATA operations are transmitted without loss even on a congested link.

For forward service provider applications, the API offers specific interfaces, by which the application can notify the API when processing steps complete or when the state of service production changes. The API uses this information to generate the required notifications and transmit them to the service user and to update the data structures needed for status reporting.

As an example, table 4-1 shows the production events that are supported by the API for the Forward CLTU service, the notifications that are sent to the service user, and the status information updated according to the parameters passed by the application. Reference [17] contains a similar table for the Forward Space Packet service.

NOTES

1 The notifications and status parameters identified in table 4-1 are defined in the Recommended Standard for the Forward CLTU Service, reference [8].

2 Although the table lists several notifications for some of the events, at most one notification is sent per event. The notification that must be generated is derived from additional arguments passed by the application.

3 The API offers generation and transmission of notifications as an optional service. Applications can opt to send ASYNC-NOTIFY invocations using the standard interfaces for transmission of operation invocations.

Table 4-1: Reporting of Production Events for the Forward CLTU Service

Event Reported	Notification Sent	Status Information Updated
Radiation of a CLTU started	none	CLTU identification last processed radiation start time CLTU status number of CLTUs processed available buffer size
Radiation of a CLTU completed.	'CLTU radiated'	CLTU identification last OK radiation stop time CLTU status number of CLTUs radiated
Radiation of a CLTU could not be started because the latest radiation time expired or the production status was interrupted.	One of: 'SLDU expired' 'unable to process'	CLTU identification last processed radiation start time CLTU status number of CLTUs processed available buffer size
Radiation of a CLTU started but could not be completed, because the production was interrupted or halted.	One of: 'unable to process' 'production halted'	CLTU status available buffer size production status
The production status changed without affecting a CLTU being radiated	One of: 'unable to process' 'production halted' 'production operational'	available buffer size production status
The uplink status changed	none	uplink status
CLTU buffer empty	'buffer empty'	available buffer size
Event actions completed	'action list completed'	none
Event actions could not be completed because of a failure	'action list not completed'	none
Event condition evaluated to false	'event condition evaluated to false'	none

4.8 LOGGING AND DIAGNOSTICS

The API Recommended Standard specifies a simple interface by which API components can pass log messages to the application and alert the application of specific events, such as an access violation identified by the API. With a few exceptions related to security alarms, the Recommended Standard does not prescribe what events shall be reported via this interface. It does, however, require conforming implementations to document the messages reported.

As an implementation option, the API Recommended Standard specifies interfaces to start and stop diagnostic traces in the API and to pass trace records to the application. As for log messages, the SLE API Recommended Standard does not prescribe the detailed contents of trace messages. It defines four trace levels and provides general guidelines of what should be traced on each of the levels. The trace levels identified by the Recommended Standard are:

a) 'Low' – state changes are traced. The information includes the old state, the new state, and the event that caused the state change.

b) 'Medium' – the trace additionally includes the type of all PDUs processed as well as additional interactions between components.

c) 'High' – the trace additionally contains a printout of all parameters of the PDU processed.

d) 'Full' – the trace additionally contains a dump of the encoded data sent to and received from the network.

Tracing can be requested for all active service instances and associations, or selectively for individual service instances.

4.9 MISCELLANEOUS SUPPORT FEATURES

The API Recommended Standard specifies a small number of utility interfaces providing specific support for SLE specific data structures. These services are required for the API itself but are also available to applications. Examples for such services are:

a) a service to process the service instance identifier, providing methods to check the validity of a service instance identifier and to translate between various formats defined for the service instance identifier;

b) a service handling time, which specifically supports the CCSDS time code formats.

5 THE SLE API OBJECT MODEL

5.1 INTRODUCTION

The design of the SLE API makes use of objects that mirror the concepts described in the SLE Reference Model (reference [3]) and the Recommended Standards for SLE transfer services (references [5], [6], [7], [8], [9]). Application programmers, who are familiar with the SLE service concepts, will therefore find the API intuitive and easy to use.

SLE concepts used for the API object model include:

a) transfer service instances;

b) associations between a service user and a service provider;

c) transfer service operations.

The following subsections provide a brief introduction to these concepts and explain how they are used in the object model of the SLE API. Subsection 5.4 concludes the discussion of the object model by an outline discussion of how an application program uses the API.

NOTE – The descriptions of the concepts in this report aim at an intuitive understanding and are therefore informal in nature. Formal and precise definitions, based on the *Abstract Service Definition Conventions* (reference [26]) can be found in the SLE Reference Model (reference [3]).

5.2 SERVICE INSTANCES AND ASSOCIATIONS

5.2.1 OVERVIEW

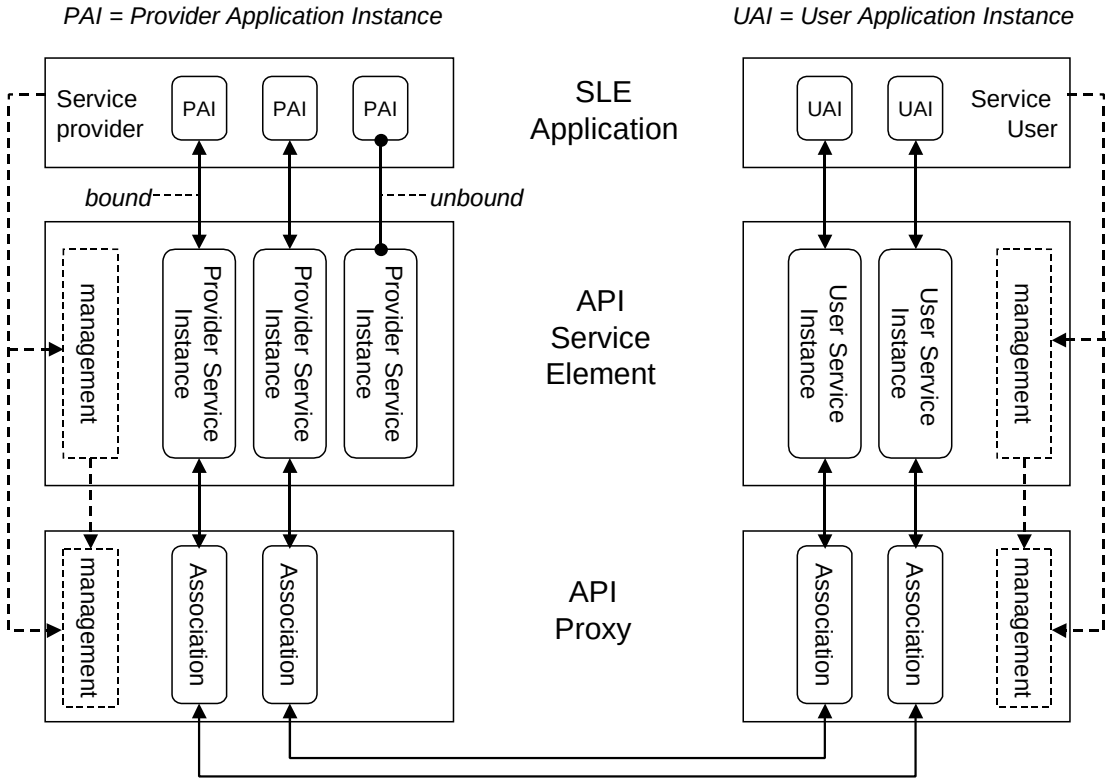
CCSDS Space Link Extension services require that every instance of transfer service provisioning be fully specified and scheduled by management in advance. Service instances are made available by a SLE service provider during the scheduled provision period. A SLE service user can actually use the service once or several times during this period. For every instance of use, the SLE service user establishes an association with the SLE service provider, by means of the BIND operation.

NOTES

1 Definition and scheduling of SLE transfer service instances is defined by the SLE Service Management Specification (reference [4]).

2 For some return link services, CCSDS Recommended Standards include the concept of ‘provider initiated binding’ in addition to ‘user initiated binding’. In this approach, the provider establishes the association to the user at start of the scheduled provision period. When this approach is applied, the roles of service user and the service provider with respect to association establishment are interchanged.

In the SLE API, these concepts are reflected by service instance objects managed by the API Service Element and by association objects managed by the API Proxy, as shown in figure 5-1. The API Recommended Standard does not prescribe how an application handles service instances, but it assumes that the application also provides a specific object for every service instance. These objects are displayed as 'User Application Instances' (UAI) and 'Provider Application Instances' (PAI). As shown in the figure, the API Service Element and the API Proxy additionally provide management interfaces to the application for configuration, start-up, and shutdown.



NOTE – The figure shows an example of three service instances scheduled by a service provider, two of which are bound to user service instances.

Figure 5-1: SLE API Object Model

A service instance object on a provider system can be accessed by exactly one SLE service user. However, a SLE service provider can use different service instance objects for concurrent use by different SLE service users. A SLE service user can create several service instance objects to communicate with different SLE service providers concurrently. The number of service instances is not constrained by the API Recommended Standard but can be constrained by an implementation of the SLE API.

5.2.2 SERVICE INSTANCES

5.2.2.1 General

The API Service Element provides interfaces to create service instance objects of a specified SLE service type and to configure these objects using the specification of the service instance supplied by management. Different classes of service instance objects are provided for support of a SLE service user application and a SLE service provider application. Objects of these two classes provide the same basic services and the same set of interfaces but differ in their detailed behavior.

All types of service instance objects provide an interface to the application to invoke SLE operations, which the peer system shall perform. The API transmits these invocations to the performer and delivers the operation return to the application, if the operation is confirmed. Service instance objects deliver operation invocations received from the peer system to the application and provide interfaces for the application to transmit the return for confirmed operations.

5.2.2.2 User Service Instances

SLE user applications create user service instance objects as needed. After creation and configuration, an application invokes the BIND operation on the service instance. The API will then attempt to establish an association to the service provider and report the result back to the application. If the BIND operation succeeds, the state of the service instance is set to 'bound' and the application can invoke further operations as needed. The application terminates use of the service by invoking the UNBIND operation on the service instance. It can then either delete the service instance or re-bind it at a later stage.

5.2.2.3 Provider Service Instances

A SLE service provider application is expected to create a provider service instance object before start of the scheduled provisioning period. Within the provision period, the API accepts a BIND invocation for a service instance, verifies that the request is legal according to the protocol and consistent with the configuration of the service instance, and forwards it to the application. If the BIND invocation is accepted by the application, the API completes the association establishment procedure and sets the state of the service instance to ‘bound’. Service provisioning operations are now exchanged until the association is terminated by the UNBIND operation or due to an abort. At the end of the scheduled provisioning period, the application is expected to delete the service instance.

Provider service instances perform SLE operations related to service parameter access and status reporting autonomously, offloading the application from these tasks.

5.2.3 ASSOCIATION OBJECTS

Association objects in the API Proxy are responsible for establishment of a data communication association between the SLE service user and the SLE service provider and for transfer of SLE protocol data units. Conceptually, the association object is created as part of the BIND operation and deleted after completion of the UNBIND operation, but an implementation has some degree of freedom in controlling the lifetime of association objects.

During periods in which a SLE service user and a SLE service provider communicate, a service instance object is linked with exactly one association object in the API Proxy. Because a SLE service user can re-bind to a service instance following an UNBIND operation or an abort, a service instance might be linked with several different association objects sequentially during its lifetime.

Association objects are not visible to SLE applications; they are used exclusively through service instance objects provided by the API Service Element.

5.3 OPERATIONS

Transfer services are defined in terms of operations that are requested and performed in the context of a scheduled transfer service instance. Operations are either invoked by the service user and performed by the service provider, or are invoked by the provider and performed by the user. SLE operations can be confirmed, i.e., the result of the operation is returned to the invoker, or unconfirmed.

As mentioned earlier (see 2.1) SLE transfer services are oriented towards a messaging paradigm. SLE operations are invoked by sending an invocation PDU from the invoker to the performer. Confirmed SLE operations are terminated by transmission of a return PDU from the performer to the invoker. Transfer of the invocation PDU and transfer of the return PDU are independent actions. A return PDU is associated with an invocation PDU by an invocation identifier.

In the SLE API, these concepts are modeled by SLE operation objects. SLE operation objects provide storage for all parameters defined for a specific SLE operation and provide implementation independent access for reading and writing of these parameters. Processing of a confirmed operation object is illustrated in figure 5-2.

On the invoker side, an operation object is initially created by the application, which fills in the invocation parameters of the operation. The operation object is then passed through the layers of the API, down to the API Proxy, which reads the parameters, constructs the PDU as required by the technology and transmits it to the peer proxy.

On the performer side, the receiving proxy creates the operation object and fills in the invocation parameters extracted from the PDU. The operation object is then passed through the layers of the API up to the application, which reads the parameters from the object and performs the operation.

If the operation is confirmed, the application stores the return parameters to the operation object and passes it back to the proxy via the service element. The proxy encodes the return PDU and transmits it to the peer proxy. On the invoker side the proxy stores the return parameters to the operation object and passes it back to the service element, which finally returns it to the application.

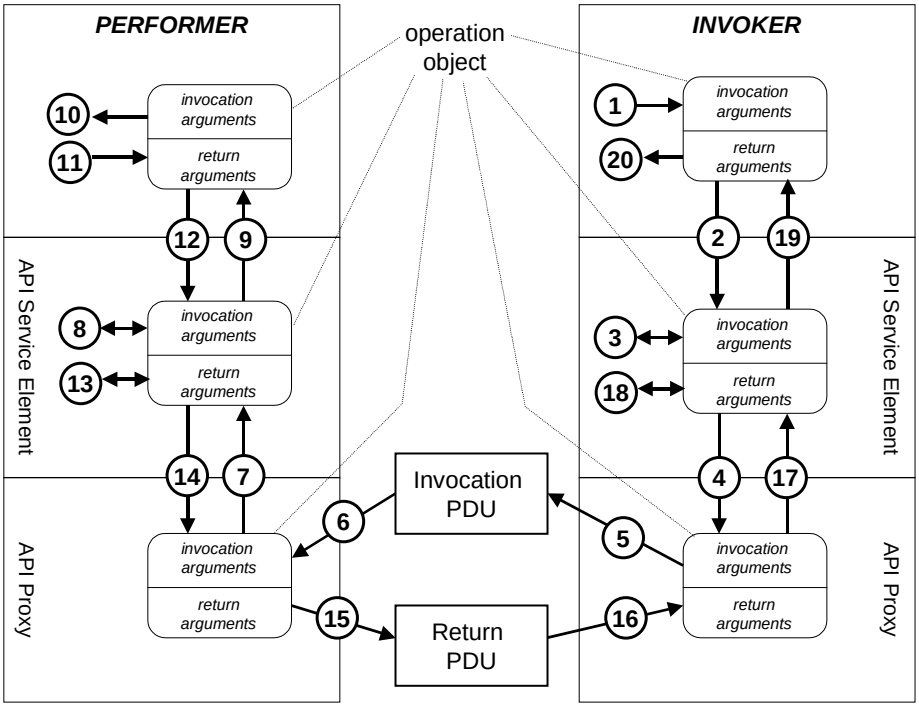
Modeling SLE operations as independent objects has a number of advantages, including the following:

a) data can be passed between API layers without copies; only references to the operation object are passed;

b) interfaces related to service provisioning can be made simple and can be identical for all service types, because only references to operation objects must be passed across these interfaces;

c) complex data structures are encapsulated by the operation objects, providing storage independent access to the application and to other API components at the level of individual parameters and simple types;

d) memory management for operation parameters can be handled by the operation objects reducing the probability of errors.



NOTE – See legend on the next page.

Figure 5-2: Processing of Confirmed Operation Objects

Legend to figure 5-2:

1. *Create an operation object and fill in invocation arguments*
2. *Pass the operation object to the service element*
3. *Check validity of the operation, verify invocation arguments, and add invocation arguments handled by the API (e.g., the invocation identifier)*
4. *If all checks pass, forward the operation object to the proxy*
5. *Read invocation arguments, format an invocation PDU as required by the technology used, and transmit it to the performer*
6. *Decode the invocation PDU, create the appropriate operation object, and fill in the invocation arguments*
7. *Pass the operation object to the service element*
8. *Check validity of the operation and verify invocation arguments*
9. *If all checks pass, forward the operation object to the application*
10. *Read the invocation arguments and perform the operation*
11. *Store the result and other return arguments to the operation object*
12. *Pass the operation object to the service element*
13. *Check validity of the operation and verify return arguments*
14. *If all checks pass, forward the operation object to the proxy*
15. *Read return arguments, format a return PDU as required by the technology used, and transmit it to the invoker*
16. *Decode the return PDU, locate the matching operation object, and fill in the return arguments*
17. *Pass the operation object to the service element*
18. *Check validity of the operation and verify return arguments*
19. *If all checks pass, forward the operation object to the proxy*
20. *Retrieve and analyze the result and possibly other return arguments*

5.4 USING THE SLE API

Ease of use has been a prime objective of the API specification. An application program must only implement a small set of interfaces, by which the API can

a) pass operation invocations and returns received from the peer system;

b) notify the application of specific events, such as breakdown of the data communications connection;

c) pass log messages for entry in the system log.

Before using the API, the application must create and configure API components described in 6.3. Following initialization, a user application typically performs the following steps:

a) Request the service element to create a service instance object for a specified service type, passing it the application interface for acceptance of events.

b) Configure the service instance by specification of a few configuration parameters, such as a logical identifier of the port at which the service provider makes the service

available. The required configuration parameters must be obtained from SLE service management, by means outside the scope of the API Recommended Standard.

c) Invoke the BIND operation using the appropriate interface provided by the service instance.

d) Invoke other service operations via the service instance as needed and accept operation invocations and returns from the API.

e) Invoke the UNBIND operation.

f) Instruct the service element to delete the service instance.

Because the SLE API implements the state tables defined by SLE Recommended Standards for transfer services, and verifies validity of the operation arguments, the need for error checking by the application is minimized.

Invocation of a confirmed operation includes the following processing steps, which are identical for the service user and service provider:

a) Request the service instance object to create an operation object of a specified type. The parameters of the operation object will be initialized by the service instance as far as possible.

b) Pass the remaining invocation parameters to the operation object, using the methods of the operation object interface.

c) Request the service instance to transmit the invocation, passing it the operation object.

d) When the operation return arrives from the peer system, the API will enter the return arguments to the operation object and return it using the appropriate method of the application interface.

When receiving an operation invocation, the service instance invokes the appropriate method of the application interface and forwards the operation object holding the invocation parameters. The application is then expected to perform the operation. If the operation is confirmed, the application must store the result and possibly other return arguments to the operation object and return it to the API.

Use of the API by a provider application follows essentially the same lines. Differences originate from the requirement that the provider application must be prepared to accept incoming BIND invocations during the scheduled service provision period. This implies that a service instance must be created and configured some time before start of the provision period. When the API receives a valid BIND invocation for the service instance within the provision period it invokes the appropriate method of the application interface. Operation invocations and returns can then be exchanged as needed. The API informs the application when the provision period ends.

6 SLE API SOFTWARE ARCHITECTURE

6.1 OBJECTIVES

The concept to enable interoperability by supply and integration of the technology specific API Proxy components, or by building a gateway form existing API Proxy components, calls for substitutability of individual API components.

For economic reasons it should be possible for developers of a SLE application (or of a gateway) to integrate API components which have been independently developed by different teams, with no or only minimal support from the development team. Delivery and integration of components can be simplified if it does not require delivery of source code and of the associated build procedures to generate executable systems in the target environment. Therefore, the API Recommended Practice documents intend to enable delivery and integration of binary libraries.

Because SLE API components must be integrated into various different deployment environments, there is a need to customize these components for use in a specific environment.

As applications generally operate on a variety of platforms, enabling portability of SLE API implementations has also been an important goal.

To enable substitutability, integration of binary libraries, and portability, the API Recommended Standard

a) defines API components with clear responsibilities and well defined behavior;

b) applies conventions for minimizing dependencies between components; and

c) details the interfaces between components.

Within these constraints, the SLE API Recommended Practice documents strive to provide as much freedom to implementers as possible.

6.2 SUBSTITUTABILITY THROUGH COMPONENT BASED DESIGN

The technology enabling independent deployment and assembly of prefabricated software components has become known as component based design.

NOTE – A detailed discussion of this technology is beyond the scope of this report. An excellent overview of the concepts and the current state of the art can be found in reference [20].

This technology has evolved only recently and to date no commonly agreed, platform independent component model is available. Component architectures that have received wider support include:

a) Microsoft's Component Object Model (COM, see reference [19]) and Distributed COM (DCOM) for communication between components in different processes or on different network nodes;

b) Java Beans or the extended version, Enterprise Java Beans (EJB).

In addition, the Object Management Group (OMG) is developing a standard for a CORBA Component Model that is supposed to integrate EJB.

Because EJB is restricted to Java, use of this component environment was not considered suitable for the API. COM defines a binary interface standard that is independent of a specific programming language as well as a set of design conventions for components. In theory, COM components can be implemented in any programming language, although that task could become rather cumbersome for some languages. For C++ and Java, and to a lesser extent for C, adherence to COM conventions is reasonably straightforward. However, COM requires presence of a runtime system, supporting dynamic runtime loading and linking of components. Because the COM runtime is not readily available on platforms other than Windows, a dependency on this system was not considered acceptable for the SLE API.

In conclusion, the following guidelines were established for the specification of the SLE API:

a) The API should be specified using the C++ language as specified by reference [18] and applying an object-oriented approach.

b) The API specification should adopt COM conventions to the extent necessary to support substitutability of API components. However, dependency on any specific software product or tool beyond a standard C++ compiler should be avoided.

The design conventions adopted from COM for implementation of API components and for the interfaces the application must provide to the API, have been dubbed the SLE ‘Simple Component Model’ (SCM). Essential conventions specified by SCM include:

a) Objects interact only via interfaces. An interface is a collection of semantically related functions providing access to the services of an object. In C++, interfaces are specified by classes containing only public, pure virtual member functions. Interfaces can be derived from other interfaces.

b) Objects can implement more than one interface and support navigation between these interfaces. Interfaces are identified by a Globally Unique Identifier (GUID) assigned to every interface specification.

c) Interfaces are considered immutable once they have been published. If modifications must be applied, a new interface with a new identifier is created.

d) The lifetime of objects is controlled by reference counting. Methods to add a reference and to release a reference are provided by every interface.

NOTE – Reference counting is not directly related to support of components. In languages that do not provide automatic garbage collection, such as C++, it is needed to enable safe memory management in the presence of multiple threads of control.

Components are software modules, which implement one or more objects and provide means by which other components can obtain references to object interfaces. Objects implemented by components are never directly exposed to other components, which only obtain references to interfaces. These conventions have the effect that client software is effectively decoupled from the internal object structure of components, and only depends on the interfaces and the behavior specified by the SLE API.

It must be stressed that components developed according to SCM do not conform to COM. However, it is possible to develop and use SLE API components in a COM environment. It is also possible to write very simple COM conforming wrappers for components conforming to SCM. Important differences to COM are:

a) Dynamic loading and linking of component servers at runtime is not supported. All components must have been linked with the program using them. Therefore, the SLE API does not require the COM library, the Registry, or any other special runtime environment.

b) Instead of supporting COM 'component servers' and 'class factories', objects within the SLE API are created via specific 'factory interfaces' defined as part of the API specification.

c) All interactions between API components are local to one address space. Use of remote procedure calls between components is not foreseen.

d) No COM defined interfaces are supported beyond the basic interface `IUnknown`, which defines the methods for navigation between interfaces and for reference counting and `IMalloc` for memory management.

For customization of API components for a specific deployment environment, the API specification takes a more traditional approach by defining a set of configuration parameters, which could for example be defined in a text file.

6.3 SLE API COMPONENTS

In response to the requirements outlined in 6.1, the SLE API Recommended Practice document defines the four components shown in figure 6-1:

a) The component API Proxy implements the features described in 4.2 and 4.3, including the association objects described in 5.2.3.

b) The component API Service Element implements the features described in 4.4 to 4.7, including the service instance objects described in 5.2.2.

c) The component SLE Operations implements the operation objects addressed in 5.3. It is defined as a separate component, because operation objects are passed between the application, the API Service Element, and the API Proxy.

d) The component SLE Utilities provides a small set of utility objects, e.g., for handling of CCSDS time codes or of SLE service instance identifiers.

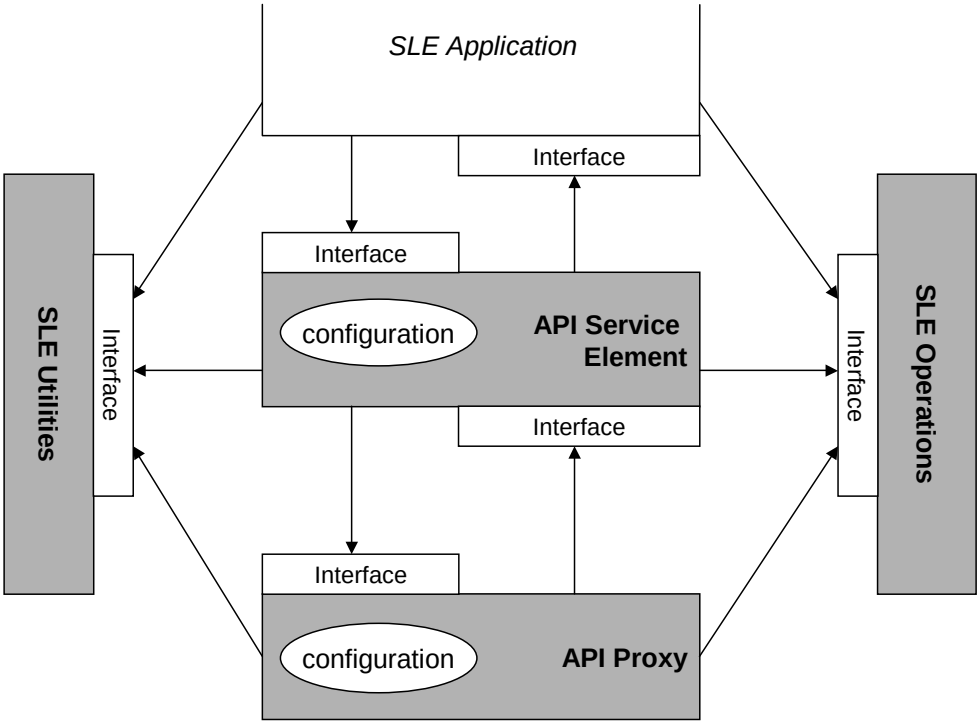


Figure 6-1: SLE API Components

The SLE Application is not a SLE API component, but the client of the SLE API. However, the SLE API Recommended Standard defines some interfaces that an application must provide for use by API components.

The services of an API component are accessible only via the interfaces defined in the SLE API Recommended Practice documents. These Recommended Standards also describe the externally visible behavior that must be associated with these interfaces. They do not define, how the components are designed and implemented.

6.4 CUSTOMIZATION

Because SLE API components must be integrated into various different deployment environments, there is a need to customize the components API Proxy and API Service Element.

NOTE – The functionality of the components SLE Operations and SLE Utilities is fully specified. Therefore, these components do not require customization.

Examples of items that must be configurable include the access control lists mentioned earlier, mapping of logical port identifiers to physical addresses, and parameters associated with the data communications system used by the proxy. In order to support substitutability of components, customization should not involve the application software or any of the other API components.

Therefore, the API Recommended Standard specifies a configuration database for the API Proxy and the API Service Element. With the exception of a few mandatory items, the Recommended Standard does not prescribe the contents of the database. It neither defines the structure and implementation of the database. The configuration database might consist of one or more text files or might be make use of directory systems or some management database. However, the Recommended Standard does require that an implementation document the contents of the database and the procedures by which the configuration parameters can be entered and updated. Definition and update of the configuration database is considered an off-line maintenance activity. Components read the configuration database at start-up of the API on request of the application.

This approach provides full freedom for developers with respect to the degree of configurability they wish to provide for an implementation. At the same time, it avoids the need to modify the application program for supply of configuration parameters when a new component is integrated, because all parameters can be specified to suit the deployment environment by editing the configuration database.

6.5 FLOWS OF CONTROL

In order to ensure substitutability, handling of multiple flows of control must be well defined at interfaces between components. The SLE API Recommended Standard defines two behaviors:

a) sequential behavior, in which a single flow of control at a time may pass an interface;

b) concurrent behavior, in which multiple flows of control can pass an interface concurrently.

NOTES

1 Multiple flows of control are frequently implemented by in process threads but can also be provided by interrupt handlers or other operating system features. In the API Recommended Standard the term ‘thread’ is used in a broader sense referring to any kind of flow of control.

2 The terms ‘sequential’ and ‘concurrent’ have been adopted from the characteristics defined in UML for operations. However, the meaning of ‘sequential’ is slightly more restrictive in the API Recommended Standard than in UML and the term ‘concurrent’ as used in the Recommended Standard actually maps to ‘concurrent or guarded’ in UML.

The selected behavior must be respected by the supplier of an interface and by the client of an interface. The same behavior is assumed for complementary interfaces, i.e., an application intending to use an API implementation supporting concurrent behavior must implement the interfaces provided to the API in a multi-thread safe manner. API components are required to support at least one of these behaviors but can support both.

For the sequential interface behavior, the API Recommended Standard defines interfaces by which the client offers means for components to wait on external events and to handle timers. Components providing concurrent behavior are expected to handle external events and timers internally.

6.6 DISTRIBUTION ASPECTS

All interfaces of SLE API components are defined in the scope of a single address space, which implies that an instance of the API is constrained to one process. An instance of the SLE API is able to handle several service instances concurrently within one process. However, applications might want to use a separate process for every service instance or for groups of service instances. Such configurations require that protocol data units for a specific service instance be routed to the correct process. Availability of such features depends on the capabilities of the data communications service, the operating system, and the implementation of the component API Proxy.

NOTES

1 The following discussion is only relevant for applications that need to respond to a BIND invocation. Data communication technologies and operating systems generally do not impose constraints for outgoing calls.

2 Although the SLE API Recommended Standard constrains interactions between API components to a single address space, it does not require that individual API components be constrained to a single process. In particular the component API Proxy might be might be implemented using objects that are distributed to more than one process.

In order to receive a BIND invocation, a responding proxy must listen on a port of the data communication service. The identifier of this port is part of the service instance configuration and made known to the initiating proxy when the BIND operation is invoked. The initiating proxy maps the identifier to technology specific address information, enabling the data communication service to deliver the BIND invocation at that port.

NOTES

1 The meaning of the term ‘port’ depends on the technology. Examples include a TCP socket on which the proxy listens for connect requests, identified by an IP address and a port number, and an OSI Session Service Access Point (S-SAP) identified by an OSI session address.

2 Depending on the technology, the port on which the proxy listens for BIND invocations may or may not be the same as the one used for actual communications. For instance, TCP requires creation and use of a new port for every connection that is accepted. In the context of this discussion, the port on which the proxy receives the BIND invocation (or the connect-request, if the BIND invocation is transmitted over an established connection) is the important one.

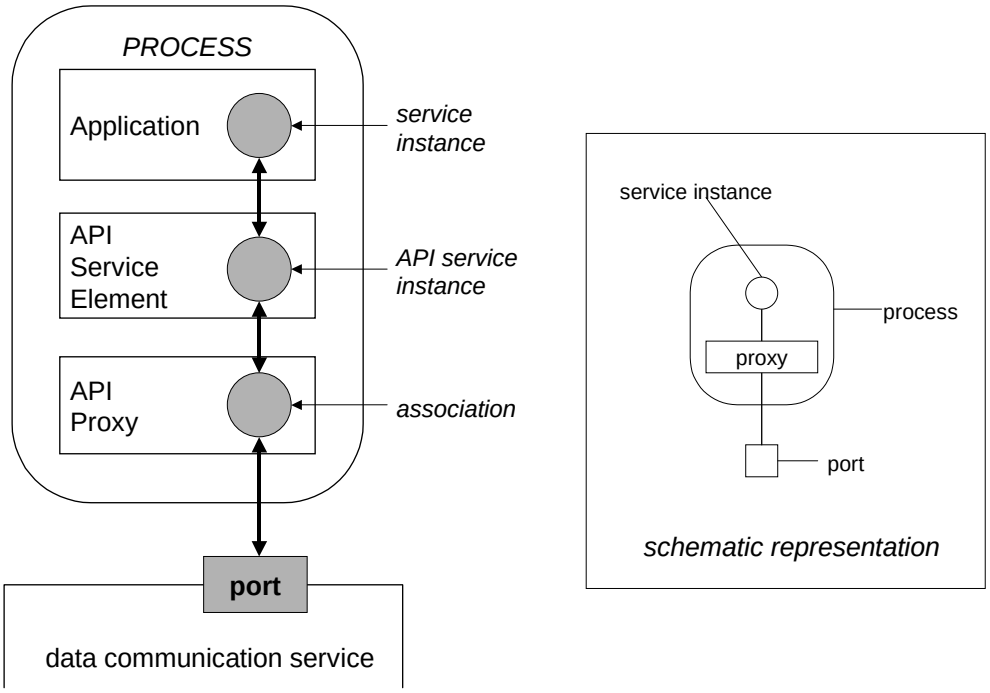


Figure 6-2: Ports, Associations, and Service Instances in the API

When receiving a BIND invocation, the API Proxy creates a new association object and then calls the API Service Element to locate the service instance. If service instances are distributed to more than one process, the BIND invocation (or the connect request) must be routed to the proxy in the process handling the service instance. This routing can be performed by the data communications service or by specific infrastructure provided by the proxy.

The essential aspects of service instance distribution are depicted on the left hand side of figure 6-2. The right hand side of the figure presents a schematic representation of this architecture, which is used to explain configurations required by the SLE API Recommended Standard.

The configurations shown in figure 6-3 must be supported by a conforming implementation of the component API Proxy. However, implementations can constrain the number of service instances (associations) and ports they are able to support. Because the SLE API Recommended Standard does not exclude constraining this number to one, some of the configurations might not be possible in practice.

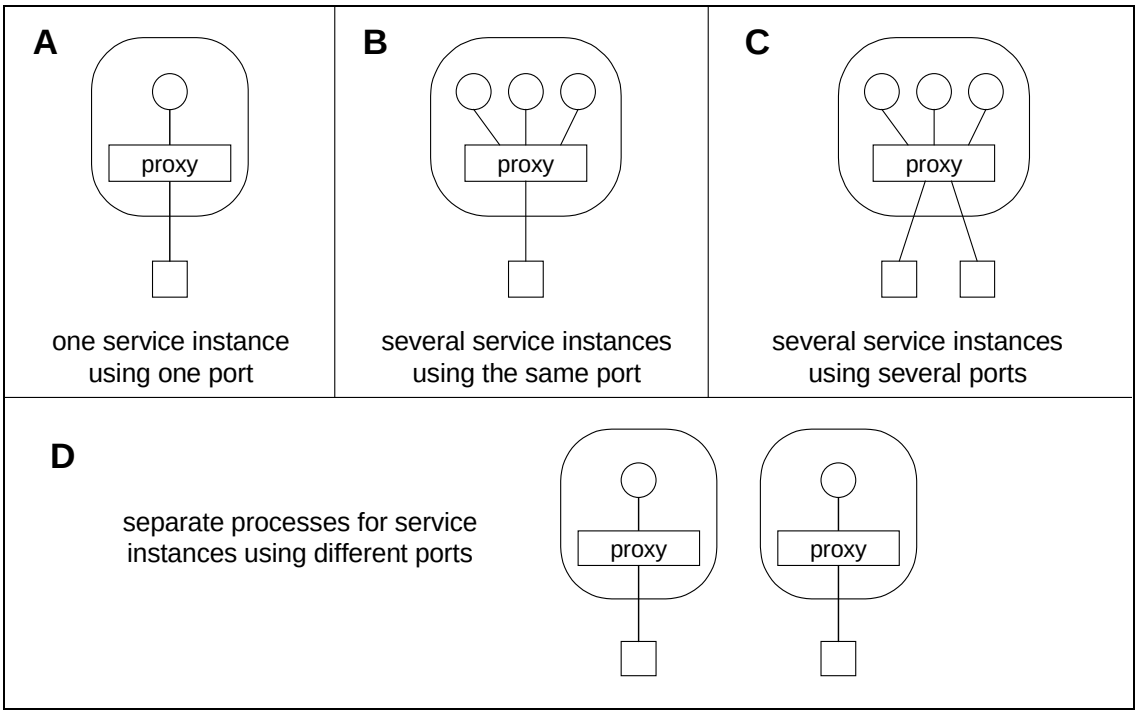


Figure 6-3: Basic Configurations of Processes and Service Instances

In configuration A, a single service instance using a single port is handled by one process. This is the bare minimum required for any useful implementation. In configuration B, one process handles several service instances for which BIND invocations are received at the same port. Configuration C shows the case where one process handles several service instances and the proxy is able to listen for BIND invocations on several ports.

It is, of course, possible to start several processes for different service instances, where each process uses one of the configurations A, B, or C (the figure only shows processes using configuration A). However, such a configuration requires that the ports used by different processes are distinct and have different addresses. This implies that each of the ports must have a distinct port identifier, and that SLE service management is able to assign the correct port identifier to service instances.

In addition to the basic configurations shown in figure 6-3, the SLE API Recommended Standard defines advanced configurations as implementation options. These configurations are illustrated in figure 6-4.

In configuration E, BIND invocations for service instances handled by different processes can be received at a single port. The data communications service or the proxy infrastructure ensures that PDUs are routed to the process, which handles the service instance. If this configuration is supported, unconstrained configuration of processes, services instances and ports is a minor extension. Therefore, the SLE API Recommended Standard requires that implementations claiming support of configuration E are also able to handle configuration F.

NOTE – Although not required by the SLE API Recommended Standard, implementations might support distribution of the processes shown in figure 6-4 to more than one node on a local network.

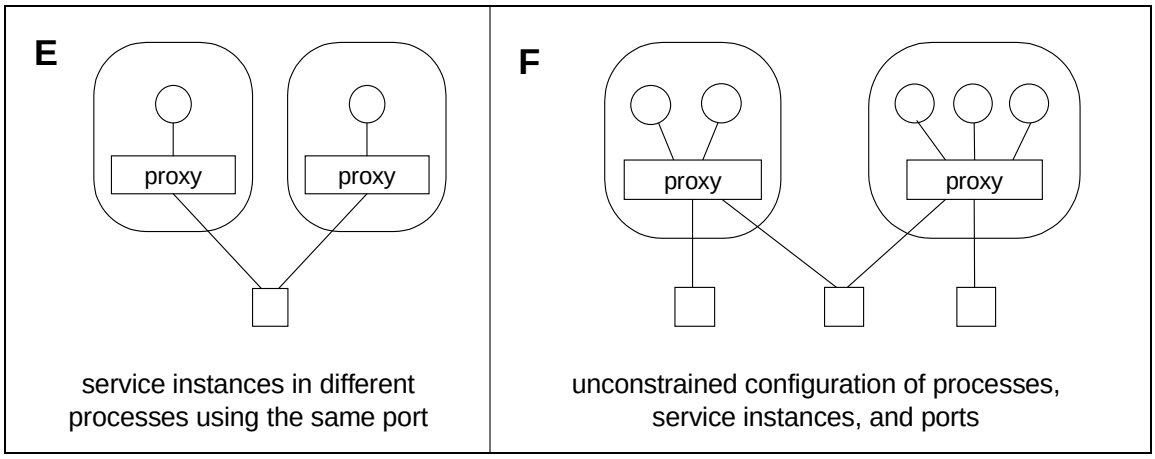


Figure 6-4: Advanced Configurations of Processes and Service Instances

6.7 PORTABILITY

The SLE API Recommended Standard does not assume any specific implementation platform. It has been the intention, to allow implementations on a wide range of different platforms.

The SLE API is initially defined for the C++ language. The design of the interfaces has anticipated development of wrappers in the C language. It has been demonstrated that such wrappers can be automatically generated from the specification by suitable generation software. Application programs that are written in languages, which support interfaces to software written in C, (e.g., Ada) can use the SLE API via a C wrapper. It should also be possible to derive interfaces for other languages with a syntax similar to C++ (e.g., Java) by minor modifications of the C++ interfaces and without the need to modify other parts of the specification.

As regards data communication services, it is the prime objective of the SLE API to support various data communication technologies by different API Proxy components. The extent, to which an API Proxy can be ported between different products for data communication services, depends on the implementation of the proxy component. It is believed that a careful design will permit encapsulating of the specifics of a given communications technology in a well-identified software module. This will significantly reduce the development effort incurred for porting of an API Proxy implementation to a different communications technology.

ANNEX A

GUIDE TO THE SLE API DOCUMENTATION

This annex provides an introduction to the SLE API documentation. It explains what information can be found in which Recommended Standard or Report and identifies the techniques that are used for the specification of the API. Annex A6 suggests what parts of the documents should be read depending on the intentions and background of readers.

A1 OVERVIEW

The SLE Application Program Interface for transfer services is specified by one Recommended Standard and six Recommended Practice documents:

a) The Recommended Standard *Space link Extension — Internet SLE Protocol* (reference [10]) specifies how the API Proxy component shall be mapped to CCSDS recognized data communication technologies. At the time of publication, CCSDS had defined a single mapping to TCP/IP and ASN.1.

b) The Recommended Practice document *Space link Extension — Application Program Interface for Transfer Services — Core Specification* (reference [10]) defines the API components addressed in 6.3 of this report and specifies the functionality and interfaces, which are common for all SLE transfer services or common for all return or for all forward services.

c) Specific APIs have been specified for several SLE transfer services. For every service, they specify the functionality and the interfaces that are specific to that service type:

   1) the Recommended Practice document *Space link Extension — Application Program Interface for the Return All Frames Service* (reference [13]) specifies the API for the RAF service defined by CCSDS;

   2) the Recommended Practice document *Space link Extension — Application Program Interface for the Return Channel Frames Service* (reference [14]) specifies the API for the RCF service defined by CCSDS;

   3) the Recommended Practice document *Space link Extension — Application Program Interface for the Return Operational Control Field Service* (reference [15]) specifies the API for the ROCF service defined by CCSDS;

   4) the Recommended Practice document *Space link Extension — Application Program Interface for the Forward CLTU Service* (reference [16]) specifies the API for the CLTU service defined by CCSDS;

   5) the Recommended Practice document *Space link Extension — Application Program Interface for the Forward Space Packet Service* (reference [17]) specifies the API for the FSP service defined by CCSDS.

NOTE – The SLE Reference Model (reference [3]) defines a large number of SLE transfer services. At the time of publication, CCSDS had not completed the Recommended Standards for all these service types. The API Recommended Practice documents for the specific SLE transfer services define an API only for those services for which transfer service specifications were available at the time of publication.

The SLE API Recommended Practice documents contain a complete specification of the interfaces, the functionality, and the externally visible behavior that must be provided by API components. Much of the detail contained in the Recommended Practice documents is only needed by developers intending to implement one or more of the API Components. Therefore, CCSDS has prepared the Report *Space link Extension — Application Program Interface for Transfer Services — Application Programmer's Guide* (reference [12]), which provides some tutorial material for software developers whishing to integrate the API into SLE user applications or SLE provider applications.

A2 PRESENTATION AND SPECIFICATION TECHNIQUES

All SLE API Recommended Practice documents contain a descriptive part and a prescriptive, or normative part.

The purpose of the descriptive part is to present a high level, yet precise description of the API elements covered by the Recommended Practice documents. For that purpose, the Recommended Practice documents describe an object model, which is presented in the Unified Modeling Language (UML).

The official definition of UML can be found in reference [21]. Because the UML specification document makes use of a formal meta-model, it requires some background on formal specification techniques. Readers not familiar with UML are therefore encouraged to refer to one of the many textbooks available on the market. References [22] and [23], published by the original authors of UML, provide in-depth coverage of the language. Reference [24] is an example of a book that provides a brief introduction to UML, sufficient to understand the models in the SLE API Recommended Practice documents. Specific conventions used for presentation of the models are explained in the core specification (reference [10]).

The prescriptive part of the API Recommended Practice documents is based on the model presented in the descriptive part and makes use of the terms explained in the models. This part is presented in a more traditional form, using numbered textual requirements. Where applicable, the specification is complemented by a set of state tables. The notation used in these state tables is derived from the notation used in UML state diagrams and is summarized in the core specification (reference [10]). A formal specification of the interfaces that must be implemented by API components and the application is provided in one or more annexes to the Recommended Practice document. These interfaces are specified in the C++ programming language as specified by reference [18].

A3 THE API CORE SPECIFICATION

The main part of the API Core Specification contains the following material:

a) specification of the overall architecture of the API, identifying the SLE API components, their responsibilities, and interrelationships;

b) for each of the four API components, detailed specifications of the functionality, the interfaces, and the externally visible behavior, as far as these are not specific for a single SLE service type;

NOTE – Features covered by the core specification include functionality needed for all SLE transfer services as well as features generally needed for return link services (e.g., handling of the transfer buffer) or for forward services (e.g., flow control for TRANSFER-DATA invocations).

c) specification of the interfaces that must be implementation by an application, for use by API components;

d) detailed state tables for the components API Proxy and API Service Element.

In addition, the Recommended Practice contains normative annexes covering the following material:

a) formal C++ specification of the common interfaces that must be implemented by API components and by application programs;

b) specification of the SLE Simple Component Model, introduced in 6.2 of this Report;

c) specification of conformance requirements for API components, including a discussion of implementation options and their constraints.

Informative annexes include selected scenarios of component configurations and interaction between API components (presented as UML diagrams) and a component interface cross reference.

A4 SLE RETURN AND FORWARD SERVICES

The SLE API Recommended Practices for return services and forward services (references [13], [14], [15], [16], and [17]) actually present a collection of independent specifications for individual service types. Due to the specific architecture of the SLE API, these specifications do not introduce new components. In general, a service type specific API specification is limited to the following:

a) specification of service type specific operation objects (see 5.3 of this Report);

b) specification of service type specific interfaces for configuration of provider-side service instances and for update of service parameters (see 5.2.2 and 4.5 of this Report).

Consequently, the only API components that need to be extended for support of new SLE transfer service types are the components API Service Element and SLE Operations.

For every transfer service covered, references [13], [14], [15], [16], and [17] provide the following material:

a) a description of extensions to the API model defined in the core specification; these model extensions are presented in UML;

b) detailed specification of the service type specific features and interfaces in textual form;

c) annexes containing the formal specification of the service type specific C++ interfaces.

A5 APPLICATION PROGRAMMER'S GUIDE

The application programmer's guide (reference [12]) is a Report that has been prepared specifically for software developers, intending to use the API within their applications. The Report provides an introduction to the software techniques used by the API, in particular to the Simple Component Model (SCM). It explains how to create and configure API components and discusses a number of scenarios demonstrating how an application can use the API.

The application programmer's guide provides tutorial material and does not replace the API Recommended Practice documents. It does, however, identify what parts of the API Recommended Practice documents must be consulted by application programmers.

Because the SLE API Recommended Practice documents provide some freedom to implementers of API components, the Report cannot provide all information application programmers need. Missing information should be provided by the documentation provided by the implementers.

A6 INTENDED AUDIENCE

The SLE API documentation intends to meet the needs of readers with various different intentions and background, including

– technical domain experts, wishing to understand the general concepts but not interested in implementation details;

– technical domain experts, who need to review and assess the specification in detail;

– application programmers, i.e., software developers, intending to integrate API components into application programs;

– API implementers, i.e., software developers, intending to implement one or more of the API components.

The following paragraphs identify what parts of the SLE API document are most relevant for each category of readers and provide suggestions concerning the sequence in which the documents should be read and further documents that should be consulted.

As mentioned in the introduction to this Report, all readers should be familiar with CCSDS Space Link Extension concepts and have a general understanding of SLE transfer services. It is strongly recommended that readers not familiar with these topics consult the Report on SLE Services (reference [2]) and possibly the SLE Reference Model (reference [3]) before proceeding with SLE API documents. Knowledge of at least one SLE return service (e.g., the Return All Frames service, reference [5]) and one SLE forward service (e.g., the Forward CLTU Service, reference [8]) would be very helpful in understanding API.

For **technical domain experts** only interested in the general concepts, it should be sufficient to read this Report, possibly skimming over the more detailed material presented in sections 4, 5, and 6. Readers interested in further details are encouraged to proceed with the descriptive part of the core specification followed by the model extensions for the transfer services of interest. The level of familiarity with UML needed to understand the SLE API models can be obtained from an introductory text on UML, e.g., from reference [24]. The information presented in the SLE API will provide the necessary background for a review of the detailed specifications and the state tables. The specification of the Simple Component Model and of the C++ interfaces requires some background in software development and programming languages. Although extensive C++ programming experience is not required, understanding of the interface specifications requires some familiarity with the C++ syntax.

Readers interested in the material provided by the Recommended Standard on technology mapping should first read the specification model presented in the descriptive part. Understanding of this model requires a general background on data communication services and protocols, but no detailed knowledge of the technology. However, the detailed specifications assume some familiarity with the proxy specification in reference [10] and with the TCP protocol.

Software developers should read this report as an introduction and then refer to the descriptive part of the core specification to familiarize with the API services, architecture, and interfaces. API implementers will have to study the complete model, whereas application programmers can focus on the components API Service Element and SLE Operations.

Before proceeding further with the API Documentation, it is recommended to study the Recommended Standards of the relevant SLE transfer services in detail, in order to fully understand the level of support and the behavior provided by the API components. It might be useful to cross-check the model descriptions in the core specification and in the API Recommended Practice documents for SLE transfer services during study of the transfer services.

In preparation for the design and implementation, it will be useful to read the application programmer's guide (reference [12]). While this guide was specifically prepared for application programmers, API implementers might find the material useful as well.

In order to complete the design and implementation, API implementers will have to study the complete core specification and the relevant service specific API specifications in detail. With the background information provided by the descriptive part of the Recommended Standards and the Reports, it is not necessary to read the Recommended Standards from beginning to end. Individual sections can be consulted when needed.

For application programmers, reference [12] provides a list of topics that are most relevant when using the API.

ANNEX B

GLOSSARY

This annex provides a glossary of important terms used throughout the report. The explanations provided in this annex are informative in nature. Normative definitions of these terms can be found in the SLE Reference Model (reference [3]), the CCSDS Recommended Standards for SLE transfer services (references [5], [6], [7], [8], and [9]), and the CCSDS Recommended Practice documents for the SLE API (references [10], [11], [13], [14], [15], [16], and [17]).

B1 APPLICATION PROGRAM INTERFACE (API)

An application program interface (API) is the specific method prescribed by a computer operating system or by another application program by which a programmer writing an application program can make requests of the operating system or another application.

In the context of the SLE API, the term is also used to refer to the set of services that can be requested via the interface and to the software that implements these services.

B2 API PROXY

The API Proxy is a component of the SLE API, which encapsulates all technology specific data communication interfaces. The API core specification defines the interfaces and the functionality of the proxy in a technology independent manner. Proxy components supporting specific technologies are defined in the Recommended Standard on technology mapping.

B3 API SERVICE ELEMENT

The API Service Element component of the SLE API, which implements lower level aspects of the SLE transfer service protocol as far as these are technology independent. The API Service Element uses the services and interfaces provided by the API Proxy.

B4 ASSOCIATION

An association is a cooperative relationship between an SLE service-using application entity and an SLE service-providing application entity. An association is formed by the exchange of SLE protocol control information through the use of an underlying communications service.

B5 BINDING

The act of establishing an association between a SLE service user and a SLE service provider is called binding. Following the general SLE approach, binding is specified in the form of a BIND operation, which is invoked by one entity and accepted (or rejected) by the other entity. If the underlying communication service is connection oriented, binding might include establishment of a data communications connection.

B6 INTERFACE

An interface is a collection of semantically related functions (in the sense of a programming language) providing access to the services of an object or a component. Interfaces only contain an abstract specification of functions and do not contain any data declarations or implementation details.

B7 OPERATION

An operation is a procedure or task that one entity (the invoker) can request of another (the performer) through an association. The terms invoker and performer are used to describe the interaction between two entities as the operations that constitute the service occur. Some operations are invoked by the service user and performed by the service provider, whereas others are invoked by the service provider and performed by the service user.

An example for a SLE transfer service operation is TRANSFER-DATA by which one annotated space-link data unit is passed from the invoker to the performer. For return services, TRANSFER-DATA is invoked by the service provider and performed by the service user. For forward services, TRANSFER-DATA is invoked by the service user and performed by the service provider.

Operations can be confirmed or unconfirmed. A confirmed operation is an operation that requires the performer to return a report of its outcome to the invoker, while the result of an unconfirmed operation is not reported to the invoker.

B8 SOFTWARE COMPONENT

A component is a software module providing a well-defined service via one or more interfaces. In this Report, the term is only used to refer to the API components identified in 6.3. API components must conform to a set of design and implementation conventions defined in the API core specification (reference [11]) and referred to as SLE Simple Component Model.

The following, more general definition of the term is taken from reference [20]:

> A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

B9 SERVICE INSTANCE

CCSDS Space Link Extension services require that transfer service provision be fully specified and scheduled by management in advance. A specific service scheduled by a service provider for a specified service user is called a service instance. Service instances are made available by a SLE service provider during the scheduled provision period. A SLE service user can actually use the service once or several times during this period. For every instance of use, the SLE service user establishes an association with the SLE service provider, by means of the BIND operation.

B10 SERVICE USER AND SERVICE PROVIDER

An entity that offers a service to another is called a service provider (provider). The other entity is called a service user (user). The terms user and provider are used to distinguish the roles of two interacting entities. In the SLE context, when two entities are involved in provision of a service, the entity closer to the space link is considered to be the provider of the service, and the object further from the space link is considered to be the user.

ANNEX C

ACRONYMS

This annex lists the acronyms used in this Report.

AL	Authentication Layer
API	Application Program Interface
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules (of ASN.1)
COM	Component Object Model
CCSDS	Consultative Committee for Space Data Systems
CLTU	Command Link Transmission Unit
CORBA	Common Object Request Broker Architecture
DCOM	Distributed COM
DEL	Data Encoding Layer
EJB	Enterprise Java Beans
FIPS	Federal Information Processing Standard
FSP	Forward Space Packet
GUID	Globally Unique Identifier
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISP1	Internet SLE Protocol One
ISO	International Organization for Standardization
MDOS	Mission Data Operation System
MUE	Mission User Entity
OMG	Object Management Group

OSI	Open Systems Interconnection
PDU	Protocol Data Unit
RAF	Return All Frames
RCF	Return Channel Frames
RFC	Request For Comments
SCM	Simple Component Model
SHF	Secure Hash Function
SL-DU	Space Link Data Unit
SLE	Space Link Extension
TCP	Transmission Control Protocol
TML	Transport Mapping Layer
UML	Unified Modeling Language