The Consultative Committee for Space Data Systems

Research and Development for
Space Data System Standards

CAST FLIGHT SOFTWARE AS A CCSDS ONBOARD REFERENCE ARCHITECTURE

EXPERIMENTAL SPECIFICATION

CCSDS 811.1-O-1

ORANGE BOOK

November 2021

The Consultative Committee for Space Data Systems

Research and Development for Space Data System Standards

CAST FLIGHT SOFTWARE AS A CCSDS ONBOARD REFERENCE ARCHITECTURE

EXPERIMENTAL SPECIFICATION

CCSDS 811.1-O-1

ORANGE BOOK

November 2021

AUTHORITY

Issue:	Orange Book, Issue 1
Date:	November 2021
Location:	Washington, DC, USA

This document has been approved for publication by the Consultative Committee for Space Data Systems (CCSDS). The procedure for review and authorization of CCSDS documents is detailed in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4).

This document is published and maintained by:

CCSDS Secretariat
National Aeronautics and Space Administration
Washington, DC, USA
Email: secretariat@mailman.ccsds.org

FOREWORD

This document is a CCSDS Experimental Specification for designing a flight software architecture using CCSDS Recommended Standards from different domains. It was contributed to CCSDS by China Academy of Space Technology (CAST) and Tsinghua University.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CCSDS has processes for identifying patent issues and for securing from the patent holder agreement that all licensing policies are reasonable and non-discriminatory. However, CCSDS does not have a patent law staff, and CCSDS shall not be held responsible for identifying any or all such patent rights.

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommended Standard is therefore subject to CCSDS document management and change control procedures, which are defined in the *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4). 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 sent to the CCSDS Secretariat at the email 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.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- China National Space Administration (CNSA)/People's Republic of China.
- Deutsches Zentrum für Luft- und Raumfahrt (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Federal Space Agency (FSA)/Russian Federation.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.
- UK Space Agency/United Kingdom.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Science Policy Office (BELSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
- Chinese Academy of Sciences (CAS)/China.
- China Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- Electronics and Telecommunications Research Institute (ETRI)/Korea.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
- Hellenic National Space Committee (HNSC)/Greece.
- Hellenic Space Agency (HSA)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- Mohammed Bin Rashid Space Centre (MBRSC)/United Arab Emirates.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Netherlands Space Office (NSO)/The Netherlands.
- Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- South African National Space Agency (SANSA)/Republic of South Africa.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- Swiss Space Office (SSO)/Switzerland.
- United States Geological Survey (USGS)/USA.

PREFACE

This document is a CCSDS Experimental Specification. Its Experimental status indicates that it is part of a research or development effort based on prospective requirements, and as such it is not considered a Standards Track document. Experimental Specifications are intended to demonstrate technical feasibility in anticipation of a 'hard' requirement that has not yet emerged. Experimental work may be rapidly transferred onto the Standards Track should a hard requirement emerge in the future.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 811.1-O-1	CAST Flight Software as a CCSDS Onboard Reference Architecture, Experimental Specification, Issue 1	November 2021	Original issue
EC 1	Editorial Change 1	December 2021	Restores missing row numbers in table 3-1

CONTENTS

CONTENTS (continued)

Section Page

Figure

CONTENTS (continued)

Table Page

1 INTRODUCTION

1.1 PURPOSE

The purpose of this Experimental Specification is to design and implement a Flexible and Unified Flight Software Architecture (FUHSI) of China Academy of Space Technology (CAST) to provide standardized basic service support for future spacecraft avionics systems. At the same time, it is an onboard reference architecture of Consultative Committee for Space Data Systems (CCSDS), providing the implementation and application of CCSDS Spacecraft Onboard Interface Services (SOIS) (reference [1]) and other standards in the spacecraft.

The software architecture is a comprehensive application of the standards of CCSDS SOIS, Space Link Services (SLS), Space Internetworking Services (SIS), the communication standards of European Cooperation for Space Standardization (ECSS), as well as some protocols of Internet Engineering Task Force (IETF). In the design process, it solves the problem of interfaces between other standards and SOIS, interfaces between the SOIS services, as well as interfaces between SOIS and the underlying devices. The design method and the application effect of the avionics system software architecture based on these standards are validated. CAST FUHSI has fulfilled the standardization, modularization, and reusability of the flight software while enhancing the function of the onboard avionics system. It can be used as the basic platform of the spacecraft software to improve the efficiency and reliability of the system and the software.

1.2 SCOPE

This Experimental Specification describes CAST FUHSI, including the following contents:

a) How to integrate the standards of CCSDS SOIS, SLS, and SIS, along with ECSS, and IETF in the flight software architecture; and how to set up the interface between SOIS services and other standards;

b) How to establish the functional connection among SOIS services in the flight software architecture;

c) How to connect the standard SOIS services to the specific devices in the flight software architecture;

d) How to apply SEDS in the flight software architecture;

e) The benefits of applying standards in the flight software architecture.

The CCSDS services and protocols involved in flight software architecture are listed as follows:

a) SOIS Subnetwork Packet Service (reference [2]);

b) SOIS Subnetwork Memory Access Service (reference [3]);

c) SOIS Subnetwork Synchronisation Service (reference [4]);

d) SOIS Message Transfer Service (reference [5]);

e) SOIS Device Access Service (reference [6]);

f) SOIS Device Virtualization Service (reference [7]);

g) SOIS Device Data Pooling Service (reference [8]);

h) SOIS Time Access Service (reference [9]);

i) TC Space Data Link Protocol (reference [10]);

j) AOS Space Data Link Protocol (reference [11]);

k) Space Packet Protocol (reference [12]);

l) Communications Operation Procedure-1 (reference [31]);

m) Encapsulation Protocol (reference [36]);

n) Asynchronous Message Services (reference [13]);

o) IP over CCSDS Space Links (reference [35]).

Among the services and protocols mentioned above, a)–h) are from SOIS, i)–m) are from SLS, n)–o) are from SIS. This Experimental Specification does not discuss the following CCSDS SOIS services:

a) SOIS File & Packet Store Service;

b) SOIS Device Enumeration Service;

c) SOIS Device Discovery Service;

d) SOIS Test Service.

Some ECSS standards are also used in the architecture:

a) Packet Utilization Standard (reference [14]);

b) 1553B Bus Standard (reference [15]).

And some protocols of IETF are also used in the architecture：

a) Internet Protocol (reference [18]);

b) Transmission Control Protocol (reference [19]);

c) User Datagram Protocol (UDP) (reference [20]).

1.3 DEFINITIONS AND CONVENTIONS

1.3.1 DEFINITIONS

1.3.1.1 Definitions from the Open Systems Interconnection Basic Reference Model

This Experimental Specification makes use of the following terms. The use of those terms in this Experimental Specification is to be understood in a generic sense, that is, in the sense that those terms are generally applicable to any of a variety of technologies that provide for the exchange of information between real systems. Those terms are:

a) entity;

b) service;

c) Service Access Point (SAP);

d) Service Data Unit (SDU);

e) Protocol Data Unit (PDU);

f) service user;

g) service provider;

h) application entity;

i) Application Layer.

1.3.1.2 Definitions from SOIS Recommendations

This Experimental Specification makes use of the following terms defined in SOIS recommendations (reference [1-10]). The use of those terms in this Experimental Specification is to be understood in a generic sense, that is, in the sense that those terms are generally applicable to any of a variety of technologies that provide for the exchange of information between real systems. Those terms are:

a) best effort;

b) data link;

c) data system;

d) data system address;

e) device;

f) Device Abstraction Control Procedure (DACP);

g) Device-specific Access Protocol (DAP);

h) Electronic Data Sheet (EDS);

i) functional interface;

j) heterogeneous network;

k) packet;

l) protocol ID;

m) Quality of Service (QoS);

n) reliability;

o) service class;

p) subnetwork;

q) user;

r) virtual device.

1.3.1.3 Terms Defined in This Experimental Specification

For the purposes of this document, the following definitions apply.

software component: An independent atomic software unit, configurable with the separation of the external environment, into which the functional interface, the program code, data, and internal variables, etc., are packaged.

transfer layer: A standard interface of data transmission for the upper-layer services and users, comprising the Open Systems Interconnection (OSI) Transport and Network Layers.

subnetwork layer: A unified interface that shields the difference of various underlying data links, below the transfer layer.

1.3.2 CONVENTIONS

In this document, the following convention is used to identify each bit in an N-bit field. The first bit in the field to be transmitted (i.e., the most left position in figure 1-1) is defined to be 'Bit 0'; the following bit is defined to be 'Bit 1', and so on up to 'Bit *N*-1'. When the field is used to express a binary value (such as a counter), the Most Significant Bit (MSB) is the first transmitted bit of the field, that is, 'Bit 0'.

In accordance with standard data-communications practice, data fields are often grouped into 8-bit 'words' that conform to the above convention. Throughout this Specification, such an 8-bit word is called an 'octet'.

The numbering for octets within a data structure starts with '0'.

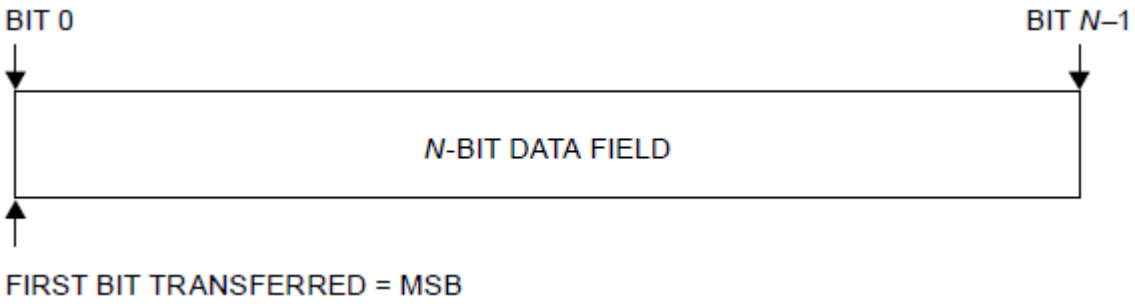

Figure 1-1: Bit Numbering Convention

1.4 DOCUMENT STRUCTURE

This document is structured as follows:

- Section 2 contains a description of CAST FUHSI and an overview of the background, software architecture, and interfaces.
- Section 3 contains the specification of the selection and integration of CCSDS and ECSS standards.
- Section 4 contains the relationship between SOIS services in the architecture.
- Section 5 contains the specification of the interfaces between SOIS services and devices in the architecture.
- Section 6 contains the application of SEDS in the architecture.
- Section 7 contains benefits of using standards in the architecture.
- Annex A contains a realization method and process example for the primitive in a CCSDS standard.
- Annex B contains the list of acronyms.
- Annex C contains description of the parameters by SEDS.
- Annex D contains description of the interfaces by SEDS.

1.5 REFERENCES

The following publications contain provisions which, through reference in this text, constitute provisions of this Experimental Specification. At the time of publication, the editions indicated were valid. All publications are subject to revision, and users of this Experimental Specification 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 publications.

[1] *Spacecraft Onboard Interface Services*. Issue 2. Report Concerning Space Data System Standards (Green Book), CCSDS 850.0-G-2. Washington, D.C.: CCSDS, December 2013.

[2] *Spacecraft Onboard Interface Services—Subnetwork Packet Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 851.0-M-1. Washington, D.C.: CCSDS, December 2009.

[3] *Spacecraft Onboard Interface Services—Subnetwork Memory Access Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 852.0-M-1. Washington, D.C.: CCSDS, December 2009.

[4] *Spacecraft Onboard Interface Services—Subnetwork Synchronisation Service*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 853.0-M-1. Washington, D.C.: CCSDS, December 2009.

[5] *Spacecraft Onboard Interface Services—Message Transfer Service*. Issue 1-S. Recommendation for Space Data System Practices (Historical), CCSDS 875.0-M-1-S. Washington, D.C.: CCSDS, (November 2012) October 2016.

[6] *Spacecraft Onboard Interface Services—Device Access Service*. Issue 1-S. Recommendation for Space Data System Practices (Historical), CCSDS 871.0-M-1-S. Washington, D.C.: CCSDS, (March 2013) October 2016.

[7] *Spacecraft Onboard Interface Services—Device Virtualization Service*. Issue 1-S. Recommendation for Space Data System Practices (Historical), CCSDS 871.2-M-1-S. Washington, D.C.: CCSDS, (March 2014) October 2016.

[8] *Spacecraft Onboard Interface Services—Device Data Pooling Service*. Issue 1-S. Recommendation for Space Data System Practices (Historical), CCSDS 871.1-M-1-S. Washington, D.C.: CCSDS, (November 2012) October 2016.

[9] *Spacecraft Onboard Interface Services—Time Access Service*. Issue 1-S. Recommendation for Space Data System Practices (Historical), CCSDS 872.0-M-1-S. Washington, D.C.: CCSDS, (January 2011) October 2016.

[10] *TC Space Data Link Protocol*. Issue 4. Recommendation for Space Data System Standards (Blue Book), CCSDS 232.0-B-4. Washington, D.C.: CCSDS, October 2021.

[11] *AOS Space Data Link Protocol*. Issue 4. Recommendation for Space Data System Standards (Blue Book), CCSDS 732.0-B-4. Washington, D.C.: CCSDS, October 2021.

[12] *Space Packet Protocol*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 133.0-B-2. Washington, D.C.: CCSDS, June 2020.

[13] *Asynchronous Message Service*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 735.1-B-1. Washington, D.C.: CCSDS, September 2011.

[14] *Space Engineering—Ground Systems and Operations—Telemetry and Telecommand Packet Utilization*. ECSS-E-70-41A. Noordwijk, The Netherlands: ECSS Secretariat, January 2003.

[15] *Space Engineering—Interface and Communication Protocol for MIL-STD-1553B Data Bus Onboard Spacecraft*. ECSS-E-ST-50-13C. Noordwijk, The Netherlands: ECSS Secretariat, 15 November 2008.

[16] *TM Synchronization and Channel Coding*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 131.0-B-3. Washington, D.C.: CCSDS, September 2017.

[17] *TC Synchronization and Channel Coding*. Issue 4. Recommendation for Space Data System Standards (Blue Book), CCSDS 231.0-B-4. Washington, D.C.: CCSDS, July 2021.

[18] S. Deering and R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460. Reston, Virginia: ISOC, December 1998.

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

[20] J. Postel. *User Datagram Protocol*. STD 6. Reston, Virginia: ISOC, August 1980.

[21] *TM Space Data Link Protocol*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 132.0-B-3. Washington, D.C.: CCSDS, October 2021.

[22] *Space Communications Protocol Specification (SCPS)—Transport Protocol (SCPS-TP)*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 714.0-B-2. Washington, D.C.: CCSDS, October 2006.

[23] *CCSDS File Delivery Protocol (CFDP)*. Issue 5. Recommendation for Space Data System Standards (Blue Book), CCSDS 727.0-B-5. Washington, D.C.: CCSDS, July 2020.

[24] *Proximity-1 Space Link Protocol—Data Link Layer*. Issue 6. Recommendation for Space Data System Standards (Blue Book), CCSDS 211.0-B-6. Washington, D.C.: CCSDS, July 2020.

[25] *Proximity-1 Space Link Protocol—Coding and Synchronization Sublayer*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 211.2-B-3. Washington, D.C.: CCSDS, October 2019.

[26] *Proximity-1 Space Link Protocol—Physical Layer*. Issue 4. Recommendation for Space Data System Standards (Blue Book), CCSDS 211.1-B-4. Washington, D.C.: CCSDS, December 2013.

[27] *Lossless Data Compression*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 121.0-B-3. Washington, D.C.: CCSDS, August 2020.

[28] *Image Data Compression*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 122.0-B-2. Washington, D.C.: CCSDS, September 2017.

[29] *Licklider Transmission Protocol (LTP) for CCSDS*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 734.1-B-1. Washington, D.C.: CCSDS, May 2015.

[30] *CCSDS Bundle Protocol Specification*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 734.2-B-1. Washington, D.C.: CCSDS, September 2015.

[31] *Communications Operation Procedure-1*. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 232.1-B-2. Washington, D.C.: CCSDS, September 2010.

[32] "Spacecraft Onboard Interface Services Electronic Data Sheets and Dictionary of Terms." Space Assigned Numbers Authority. https://sanaregistry.org/r/sois.

[33] *Electronic Data Sheets and Dictionary of Terms for Onboard Devices and Components*. Report Concerning Space Data System Standards (Green Book). Forthcoming.

[34] *Spacecraft Onboard Interface Services—XML Specification for Electronic Data Sheets*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 876.0-B-1. Washington, D.C.: CCSDS, April 2019.

[35] *IP over CCSDS Space Links*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 702.1-B-1. Washington, D.C.: CCSDS, September 2012.

[36] *Encapsulation Packet Protocol*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 133.1-B-3. Washington, D.C.: CCSDS, May 2020.

2 OVERVIEW OF CAST FLIGHT SOFTWARE ARCHITECTURE

2.1 BACKGROUND

With the development of space technology, new requirements have been put forward for the convenience and ease of operation of spacecraft. Spacecraft avionics systems are responsible for implementing those requirements, whose autonomy and ability of internetworking need to be enhanced. The efficiency and reliability of spacecraft avionics systems also need to be improved.

a) The autonomy of spacecraft should be enhanced gradually. Functions such as autonomous mission planning, self-diagnostics, and autonomous housekeeping need to be implemented by avionics systems. Additionally, the computing ability needs to be extended as required.

b) The ability of internetworking needs to be provided. The space network and onboard network should be designed in a uniform way, which will support standard protocols and isolate the influence of changes on data link and protocols to upper layers. Flexible information transfer mechanisms will be implemented to allow the cooperation of multiple spacecraft or devices inside a spacecraft. Thereby the user can focus on the implementation of algorithms to support increased autonomy.

c) The spacecraft-ground operation interfaces and onboard interfaces should be standardized, which will not only provide convenient and powerful interfaces in a standard way to the ground users, but also support the changes on onboard interfaces without affecting the upper layer applications.

The spacecraft avionic system should provide supportive services to the traditional spacecraft functions, which include telecommand management, telemetry management, housekeeping management, thermal control management, power management, etc. The development process of application software can be simplified by the integration of common services. Based on the requirements above, multiple domain requirements of spacecraft avionics systems have been analyzed by CAST, services and protocols of CCSDS and ECSS have been selected and integrated (as specified in section 3), and avionics system flight software architecture has been designed. The purposes are:

a) providing a standard software platform to support the intelligent applications for future spacecraft, the space internetworking, and onboard networking;

b) accelerating the reuse of onboard software, onboard devices, and ground test software, and in the meantime, enhancing the system functions and reducing the repetition of development;

c) transforming the development method from manual programming to assembly of software via tools based on software architecture and software components, which will improve software efficiency and system reliability.

2.2 SOFTWARE ARCHITECTURE

2.2.1 OVERVIEW

The principles of CAST FUHSI design are as follows:

a) **Layering:** A complex problem is simplified by decomposing it into several layers. CAST FUHSI is a layered architecture in which services and interfaces of each layer are standardized. On one hand, the layered architecture shields the influence of the change on the hardware interfaces and protocols from the upper layers and supports the upgrade of technology, which makes FUHSI very flexible. On the other hand, the common functions can be provided through standard services, which can increase the reusability of software.

b) **Standardization of operating system interfaces and unified framework of device drivers**: In order to support the change of operating systems, CAST FUHSI standardizes the operating system interfaces. The framework of device drivers is defined in order to support different types of device interfaces and provide the extension ability to satisfy the requirements of controlling various devices.

c) **Unified information transfer mechanism:** A unified information transfer mechanism is established based on CCSDS standards, ECSS standards, and IETF standards, which support the integrated communications and standardized design over ground-to-spacecraft, onboard, and spacecraft-to-spacecraft links. The changes and upgrades of protocols, as well as the flexible information transmission among upper layer applications, are also supported.

d) **Standardized components and their interfaces:** The standardized components and their interfaces are defined in the software architecture to provide the standard services with software components. The development of new mission software can be assembled by standard components and mission-specific components, which can promote the development process and shorten the software development cycle. Various requirements of different projects must be considered during the design of service components. The common requirements of projects shall be abstracted, and the variability shall be identified and isolated by parameters, so as to increase the flexibility and reusability of the components.

Based on the above principles, CAST FUHSI consists of an operating system layer, middleware layer, and Application Management Layer, as depicted in figure 2-1. Application Management Layer and Application Support Layer of the middleware layer constitute the Application Layer. The software architecture is established on the basis of hardware. The hardware includes various components for onboard computers, which are the operation base of the flight software. The hardware components include Central Processor Unit (CPU), Read-Only Memory (ROM), Random-Access Memory (RAM), clocks, watchdog, 1553B interface, backplane bus interface, Universal Asynchronous Receiver/Transmitter (UART) interface, Analogue (AN) interface, Memory Load (ML) interface, On/off command interface, Digital Serial (DS) interface, Time Trigged Ethernet (TTE) interface, extension interface, and so on.

The operating system layer is the supporting platform that shields the differences on hardware and operating systems through the framework of device drivers and operating system interfaces. The middleware layer, which is the core of the software architecture, contains software components to implement services and protocols from CCSDS and ECSS. The onboard communications are standardized by SOIS service components; the spacecraft-to-ground and spacecraft-to-spacecraft Data Link Layer protocols are standardized by Telecommand (TC) space data link protocol and Advanced Orbiting System (AOS) space data link protocol. Combined with Space Packet Protocol (SPP) and UDP/IP of transfer layer and Asynchronous Message Service (AMS) of the Application Support Layer, the integrated communications over spacecraft-to-ground, onboard spacecraft and spacecraft-to-spacecraft links can be implemented. With the support of the operating system layer and middleware layer, most of the functions can be implemented by the combination of common service components. Based on the architecture, users only need to select and configure the components from each layer, then develop the mission specific software and assemble the software with the components, achieving the goal of software fast-development.

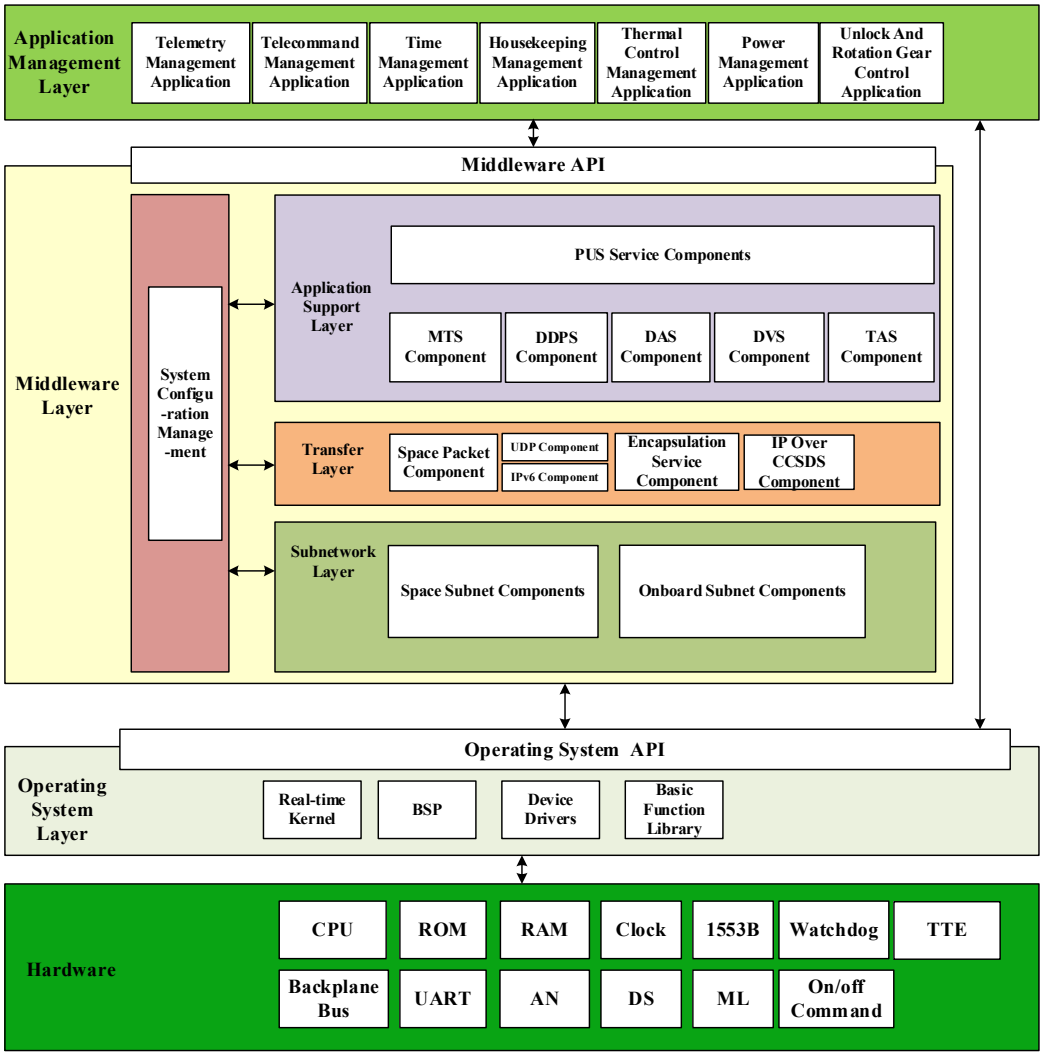

Figure 2-1: CAST Flight Software Architecture

2.2.2 OPERATING SYSTEM LAYER

The interface of Operating System is encapsulated, and a unified Application Program Interface (API) is provided by the Operating System Layer. Any operating system that is supported by this unified API can be used in the avionics system, which will allow the operating system updates. The Operating System consists of a real-time kernel, Board Support Package (BSP), device drivers, and basic function libraries. When a new hardware interface is to be supported, new device drivers can be added.

2.2.3 MIDDLEWARE LAYER

Middleware is a common service platform between the Operating System Layer and Application Management Layer, which has standard program interfaces and protocols. Middleware can provide the data exchange and cross support among different hardware and operating systems. In order to make the middleware extendable and support the upgrade of technology, the middleware is divided into three layers, with each layer being configurable through system configuration management. The layers are:

a) Subnetwork Layer. In this layer, a unified software interface is defined to shield the difference on data links. Additionally, a set of service components are provided in this layer to support the upper layer components, which include onboard subnet components and space subnet components. The onboard subnet components contain several components to implement the SOIS Packet Service, Memory Access Service, Synchronization Service, and data link convergence functions. The space subnet components consist of a TC component, an AOS component, and so on. This layer can support the add-in and change of different data link convergence components through configuration; thus the change of hardware interfaces and protocols would not influence the upper layers.

b) Transfer Layer. This layer is a combination of OSI Transport Layer and Network Layer, providing standard interfaces to the layers above for data transfer. Transport layer includes a UDP component to implement UDP protocol. Network layer includes an SPP component, IPv6 component, IP over CCSDS component, and Encapsulation Protocol component. SPP and encapsulation protocols can be distinguished by the packet version number, and they are compatible in Network Layer. The IP component can work on top of the Encapsulation Protocol component.

c) Application Support Layer. This Layer provides the standard service components to support the application, which include the SOIS Application Support Layer services and Packet Utilization Standard (PUS) services. Currently, SOIS Message Transfer Service (MTS) and AMS are implemented to support the message communications of application process. Device Access Service (DAS), Device Virtualization Service (DVS) and Device Data Pooling Service (DDPS) are provided by 3 corresponding components to support the access of devices and parameters. Time Access Service (TAS) is to support the access of onboard time. PUS service components are mainly focused on the related services in the spacecraft avionics domain, which include

telecommand verification component, device command distribution component, onboard operation scheduling component, memory management component, time management component, housekeeping & diagnostic data reporting component, onboard storage and retrieval component, onboard monitoring component, event report component, event-action component and so on.

2.2.4 APPLICATION MANAGEMENT LAYER

Application Management Layer contains most of the common functions of avionics system, which include telemetry management application, telecommand management application, housekeeping management application, time management application, thermal control management application, power management application, unlock and rotation gear control application, and so on. With the support of basic services in the lower layer, the implementation of Application Management Layer only needs to integrate the different basic services according to specific logic.

The implementation of this layer may be different among different missions. With the support of a multi-task operating system, any of several tasks or processes could use the standard interface provided by the middleware layer to accomplish the specific functions of the mission. The interface of MTS will be used for message communications among tasks or processes.

2.3 INTERFACES

2.3.1 INTERFACE OF EACH LAYER

In the software architecture, each layer provides a standard interface for the upper layer. The implementation of the protocols must conform to the requirements of the interface. The interfaces are:

a) Operating System Layer interface: including task management interface, interrupt management interface, memory management interface, semaphore management interface, timer management interface, IO interface, user support library interface, and so on.

b) Subnetwork Layer interface: including Packet Service interface, Memory Access Service interface, Synchronization Service interface, TC interface, AOS interface, and so on.

c) Transfer Layer interface: including SPP interface, UDP interface, IPv6 interface, IP over CCSDS interface, Encapsulation Protocol interface and so on.

d) Application Support Layer interface: including PUS interface, MTS interface, DDPS interface, DAS interface, DVS interface, TAS interface and so on.

2.3.2 INTERFACE OF SOFTWARE COMPONENT

The middleware of the software architecture is implemented by software components. The interface of components adopted by CAST consists of component inner parameters and an outside interface. The outside interface contains the following interface types:

a) The provided interface to the upper layers, including

   1) Initialization Interface, which can be called by other components to accomplish the initialization process,

   2) Functional Interface, which can be called by other components to accomplish the main function of the component,

   3) Configuration Interface, which can be called by system configurator to accomplish the configuration of the component;

b) The required interface from lower layers, which can be called by the component, and which can be implemented through configuration.

3 INFUSION OF SERVICES AND PROTOCOLS STANDARDS INTO CAST SOFTWARE ARCHITECTURE

3.1 GENERAL

Services and protocols of CCSDS and ECSS have been selected and integrated in CAST FUHSI as specified in 2.1, with the purpose of providing standard services, protocols, and related software components for future intelligent and internetworking applications to cover the space networks as well as the onboard networks, which can fulfill the flexible exchange of information and enhance the system functions.

Specifically, the standards integrated contain those from CCSDS SOIS domain, SLS domain, and SIS domain, as well as PUS and 1553B standard from ECSS. The steps of integration are:

a) analyzing the requirements of avionics systems serving for different types of CAST spacecraft;

b) analyzing the adaptability and applicability of CCSDS standards, ECSS standards, and IETF standards;

c) mapping the requirements to standard services and protocols to construct the avionics system service and protocol architecture.

In order to help understanding the process of selection and integration of standards as well as provide reference to the application and extension of standards, this section will focus on the following contents:

a) services and protocols architecture, including requirements analysis, analysis and selection of standard services and protocols, design of services, and protocols architecture;

b) the relationship between SOIS and other standards, including the relationship between SOIS and PUS, the relationship between SOIS and SLS protocol, and the relationship between SOIS and SIS protocol.

3.2 SERVICE AND PROTOCOL ARCHITECTURE

3.2.1 REQUIREMENTS ANALYSIS

The architectures of CCSDS and ECSS standards are both very complex, within which protocols need to be selected according to the requirements of applications. Therefore, CAST analyzed the requirements of different types of spacecraft such as remote sensing, navigation, telecommunication, crewed spaceship, deep space, and so on, and then determined the common requirements for avionics systems that are considered as the input of service and protocol architecture design.

The results of analysis show that common requirements of avionics system flight software include seven top-level functions: telecommand management, telemetry management, time management, housekeeping management, thermal control management, power management, unlock and rotation gear control, etc.

a) Telecommand management is an important way to control the spacecraft, which includes operation of telecommunication, real-time command distributing, time-tagged command distributing, providing data input channels to other application processes, etc.;

b) Telemetry management is an important way to acquire the spacecraft running state data and results of telecommand, which includes telecommand verification, acquiring device states, organizing telemetry data, data storage and retrieval, data scheduling and download, etc.

c) Time management is used to manage the synchronization of onboard time of different devices and ground system, which includes central time correction, average time correction, time distribution, etc.

d) Housekeeping management is used to provide the health management of spacecraft, which includes parameter monitoring, event report, event-action, memory management, onboard maintenance, important data storage and retrieve, self-test, system reconfiguration, etc.

e) Thermal control management includes open loop control, close loop control, failure detection and handling, thermal parameters set, etc.

f) Power management includes electricity adjust, power distribution, coulometer control, battery temperature excess protection, etc.

g) Unlock and rotation gear control includes explosive device control, antenna and solar array driving control, etc.

In addition to the common requirements mentioned above, different spacecraft have some specific requirements, such as autonomous task scheduling, autonomous navigation, routing among spacecraft, emergency return, and environment control.

Ground-spacecraft interface protocol and onboard interface protocol will be needed for the implementation of all these requirements mentioned above by avionics systems, in order to communicate with ground systems and other devices. In the meantime, some common services are needed by different functions. For example, the command sending service is needed by functions such as telecommand management, housekeeping management, thermal control management, and power management. Telemetry data acquiring service is also needed by functions such as telemetry management, housekeeping management, thermal control management, and power management. Time access service is needed by telemetry and telecommand functions. Functions related to intelligence, such as autonomous mission planning and self-determination, also need to acquire telemetry data and send commands. Additionally, different functions need a message transfer service to achieve cooperation.

3.2.2 SELECTION AND ANALYSIS OF STANDARD SERVICE AND PROTOCOL

Fundamental functions such as telemetry data acquiring, command sending, message transfer, and time access (mentioned in 3.2.1) could be implemented with CCSDS standard services and protocols.

CCSDS domains related to these functions include SOIS, SLS, and SIS. Layered architecture defined by SOIS can shield the upper layer from the influence of hardware changes and provide a set of standard onboard services to support the upper layer applications. Standards from SLS and SIS domains can provide spacecraft-to-ground and spacecraft-to-spacecraft communication services. Those domains of CCSDS focus more on the services of lower layer, but less on the direct support to top-level applications. PUS published by ECSS has defined 16 services, which has standardized the interface between ground and spacecraft in Application Layer. Additionally, the services can be combined to satisfy the top-level application functions. Thus PUS can be a valuable supplement to CCSDS. Onboard bus protocol such as 1553B interface protocol defined by ECSS could also be used together with the Subnetwork Layer services defined by CCSDS SOIS domain.

Based on the consideration above, services from CCSDS SOIS, SLS, and SIS domain can be integrated with ECSS PUS and 1553B, which will be the core of middleware in CAST FUHSI. Services and protocols from each CCSDS domain can be selected based on the following considerations.

a) CCSDS space communications protocols are developed by workgroups of SLS and SIS domain. There are five layers in the CCSDS space communications protocols reference model, including Physical Layer, Data Link Layer, Network Layer, Transport Layer, and Application Layer. The selection of protocols for each layer is as follows.

    1) CCSDS has a standard for Physical Layer called Radio Frequency and Modulation Systems, which is mainly related to hardware implementation and therefore not considered in CAST FUHSI.

    2) CCSDS defines two sublayers in the Data Link Layer: Data Link Protocol sublayer and Synchronization and Channel Coding sublayer. CCSDS has developed five protocols for the Data Link Protocol sublayer: TM Space Data Link Protocol (reference [21]), TC Space Data Link Protocol, AOS Space Data Link Protocol, Proximity-1 Space Link Protocol—Data Link Layer, and Unified Space Link Protocol (USLP). As services defined by AOS have covered all the services defined by TM, and there are no spacecraft of CAST using TM Space Data Link Protocol, AOS can be used to perform the telemetry downlink function. TC Space Data Link Protocol can be used for uplink function. In case of image and voice, uplink function is needed in a space station mission; AOS can also be used as an uplink protocol. Proximity-1 (references [24], [25], and [26]) and USLP are not implemented in the software architecture temporarily. Thus Data Link Layer protocols from SLS domain selected by CAST flight software are TC and AOS Space Data Link Protocol, together with TM Synchronization and

Channel Coding (reference [16]), TC Synchronization and Channel Coding protocol (reference [17]), and COP-1 (reference [31]).

3) There are protocols such as TCP (reference [19]), UDP (reference [20]), SCPS-TP (reference [22]), LTP (reference [29]) in the Transfer Layer and SPP, Encapsulation Protocol (reference [36]), IP protocol (reference [18]), and IP over CCSDS protocol (reference [35]) in the Network Layer. In CAST FUHSI, UDP is used in Transport Layer. SPP, Encapsulation Protocol, IP protocol, and IP over CCSDS protocol are used in the Network Layer. As these protocols are used, it is easy to support the extension of the ground network to the space network.

4) Application Layer protocols include CFDP (reference [23]), lossless data compression (reference [27]), image data compression (reference [28]), BP (reference [30]), AMS, etc. Lossless data compression and image data compression are mostly related to hardware, in addition, CFDP and BP are not used in CAST spacecraft currently. Therefore they are not included in CAST FUHSI. AMS could be used not only as a way to transfer message over space communications links, but also over onboard communications links, which can achieve the unified communications of space and onboard networks. Hence, AMS is used in CAST FUHSI.

b) CCSDS onboard communications protocols are developed by workgroups of SOIS. There are three layers in the SOIS reference architecture: the Subnetwork Layer, the Transfer Layer, and the Application Support Layer. The selection of each layer is as follows:

1) Subnetwork Layer contains Packet Service (PS), Memory Access Service (MAS), Synchronisation Service (SYNC), Device Discovery Service (DDS), Test Service (TS). PS is mainly used to transfer various packets over an onboard data link. MAS is used to access the memory or register of inside a device. SYNC can be used to provide the onboard time. Since these 3 services are fundamental services of CAST FUHSI, they are all adopted. DDS and TS can be used for device plug-and-play, which are not necessary currently. Therefore they are not adopted temporarily.

2) Transfer Layer in SOIS reference model is optional, but it is absolutely necessary in CAST FUHSI. The main concern is to syncretize the space communications and onboard communications, as well as to provide the routing mechanism among different data links, which can support remote device access, message transfer, and remote memory access between terminals on different buses. UDP, SPP, Encapsulation Protocol, IP protocol, and IP over CCSDS protocol are used in this layer. As global IPv4 address resources have been basically exhausted, and IPv6 is to be used as the next generation internet protocol, so IPv6 is selected as the network protocol both in onboard communication and space communication scene.

3) Application Support Layer in SOIS reference model contains Command and Data Acquisition Service (CDAS), TAS, MTS, File and Packet Store Service (FPSS),

and Device Enumeration Service (DES). CDAS consists of DAS, DVS, and DDPS, which are used for the device data acquiring and command sending. TAS is used to acquire the onboard time. MTS is used to communicate between different applications inside the same devices or across different devices. FPSS is used to manage files and packets. DES is used for plug-and-play. Because the first three services are related to device access, time acquire, and message share, which are the fundamental functions of CAST FUHSI, they are all adopted. As file management and plug-and-play are not involved in CAST FUHSI currently, they are not adopted.

c) As PUS protocol is the supplement of CCSDS protocol in the Application Layer, and ECSS 1553B interface protocol is the supplement of CCSDS protocol in the Data Link Layer, they are all adopted in CAST FUHSI.

3.2.3 SERVICE AND PROTOCOL ARCHITECTURE DESIGN

Through the analysis of requirements, standard services, and protocols, requirements can be mapped to services and protocols. Namely, by analyzing the way to accomplish the common functions with combinations of services and protocols, and by analyzing the way to build the relationship between different services and protocols, CAST flight software service and protocol architecture can be formed, which can be applied to remote sensing, navigation, telecommunication, crewed spaceships, and so on.

Common functions mapping to services and protocols are presented in table 3-1.

Table 3-1: Common Functions Mapping to Services and Protocols

No.	Function	SOIS Services and Protocols	SLS Services and Protocols	SIS Services and Protocols	ECSS Services and Protocols
1	Telemetry management	MTS, DDPS, DAS, DVS, TAS, PS, MAS, SYNC	TC, COP-1 (for earth orbit only), AOS, SPP, Encapsulation Protocol	AMS, UDP, IPv6, IP over CCSDS	PUS Housekeeping and diagnostic data reporting service, PUS parameter statistics reporting service, PUS Onboard storage and retrieval service, packet forwarding control service, ECSS 1553B
2	Telecommand management	MTS, DAS, DVS, TAS, PS, MAS, SYNC	TC, COP-1, AOS, SPP, Encapsulation Protocol	AMS, UDP, IPv6, IP over CCSDS	PUS Telecommand Verification Service, PUS Device command distribution service, PUS Onboard operations scheduling service, ECSS 1553B

No.	Function	SOIS Services and Protocols	SLS Services and Protocols	SIS Services and Protocols	ECSS Services and Protocols
3	Housekeeping management	MTS, DDPS, DAS, DVS, TAS, PS, MAS, SYNC	TC, COP-1, AOS, SPP, Encapsulation Protocol	AMS, UDP, IPv6, IP over CCSDS	PUS Event reporting service, PUS Onboard monitoring service, PUS event-action service, PUS Memory management service, ECSS 1553B
4	Time management	MTS, DAS, DVS, TAS, PS, MAS, SYNC	TC, COP-1, AOS, SPP, Encapsulation Protocol	AMS, UDP, IPv6, IP over CCSDS	PUS Time management service, ECSS 1553B
5	Thermal control management	MTS, DDPS, DAS, DVS, PS, MAS	TC, COP-1, AOS, SPP, Encapsulation Protocol	AMS, UDP, IPv6, IP over CCSDS	PUS Event reporting service, PUS Onboard monitoring service, PUS event-action service, PUS Device command distribution service, PUS function management service, ECSS 1553B
6	Power management	MTS, DDPS, DAS, DVS, PS, MAS	TC, COP-1, AOS, SPP, Encapsulation Protocol	AMS, UDP, IPv6, IP over CCSDS	PUS Event reporting service, PUS Onboard monitoring service, PUS event-action service, PUS Device command distribution service, PUS function management service, ECSS 1553B
7	Unlock and rotation gear control	MTS, DDPS, DAS, DVS, PS, MAS	TC, COP-1, AOS, SPP, Encapsulation Protocol	AMS, UDP, IPv6, IP over CCSDS	PUS Event reporting service, PUS Onboard monitoring service, PUS event-action service, PUS Device command distribution service, ECSS 1553B

Based on table 3-1 as well as the results of selection and analysis of standard services and protocols, the service and protocol architecture of CAST flight software is formed as shown in figure 3-1.

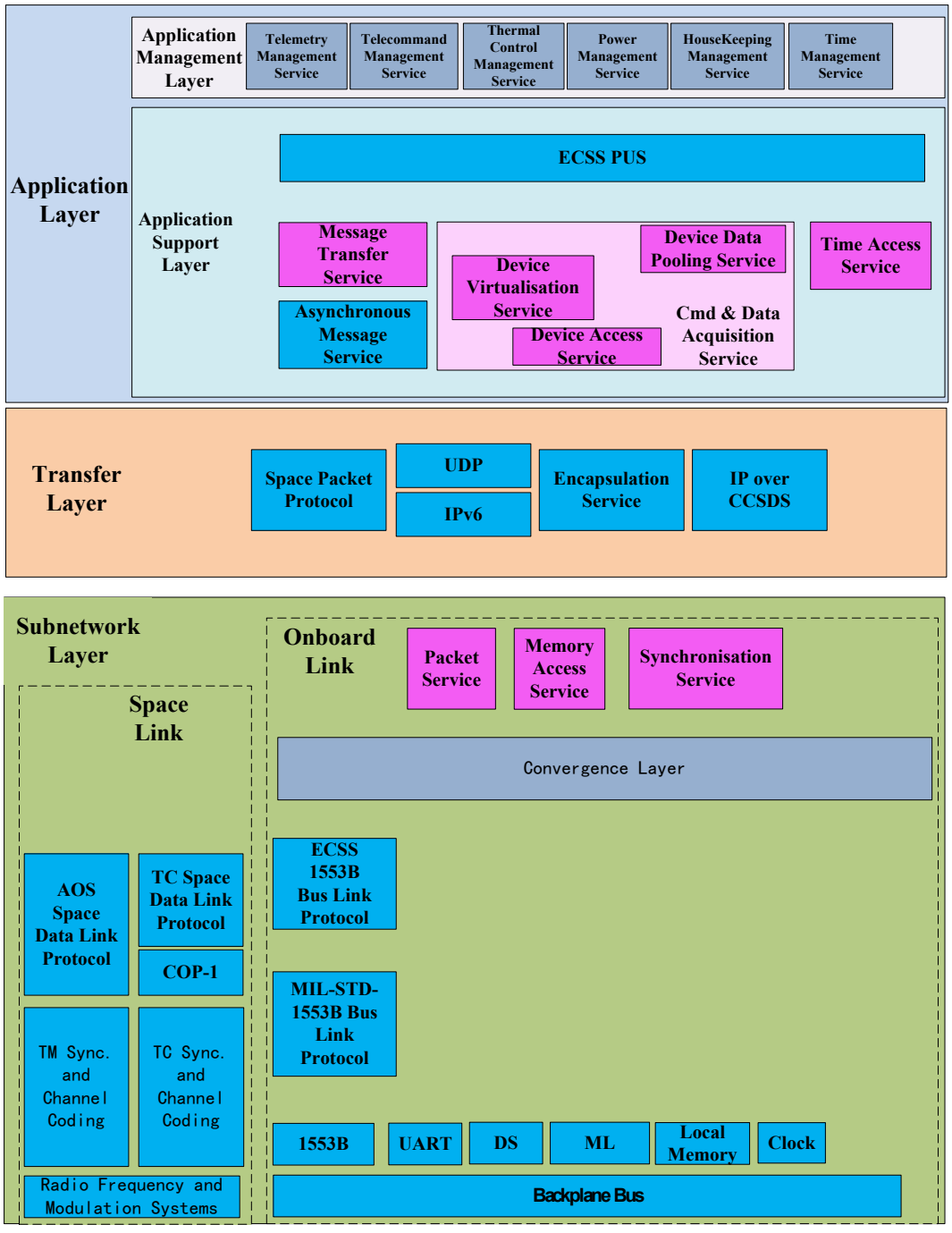

Figure 3-1: CAST Flight Software Service and Protocol Architecture

The architecture consists of 3 layers: Application Layer, Transfer Layer, and Subnetwork Layer, which are as follows:

a) Application Layer

The Application Layer consists of the Application Management Layer and the Application Support Layer. The Application Management Layer includes top level

functions such as telemetry management, telecommand management, housekeeping management, time management, thermal control management, power management, unlock and rotation gear control, etc. The functions can be provided by combining underlying layer services.

The Application Support Layer includes SOIS services such as CDAS, TAS, MTS, and standard services defined by PUS.

b) Transfer Layer

SPP is used in Transfer Layer for routing, which is extended with Source APID or Destination APID added in the secondary header of space packet. UDP and IP are also supported in this layer.

c) Subnetwork Layer

Subnetwork Layer contains space data link and onboard data link services and protocols, which can support the Transfer Layer and Application Support Layer. Space data link function is provided by TC, COP-1, and AOS. Onboard data link function is provided by PS, MAS, and SYNC. Each onboard data link can support standard subnetwork service through corresponding convergence layer protocols and data link protocols, which can shield the difference of data links. The supported data link includes 1553B, UART, ML, DS, etc. It can be easily extended to support other buses and interfaces.

The relationship between SOIS services and other standards is detailed in 3.3, the relationship between SOIS services is detailed in section 4, and the interface between SOIS services and devices is detailed in section 5.

The services and protocols in the architecture are implemented by the corresponding software components in the software architecture, the relationship is shown in figure 3-2.

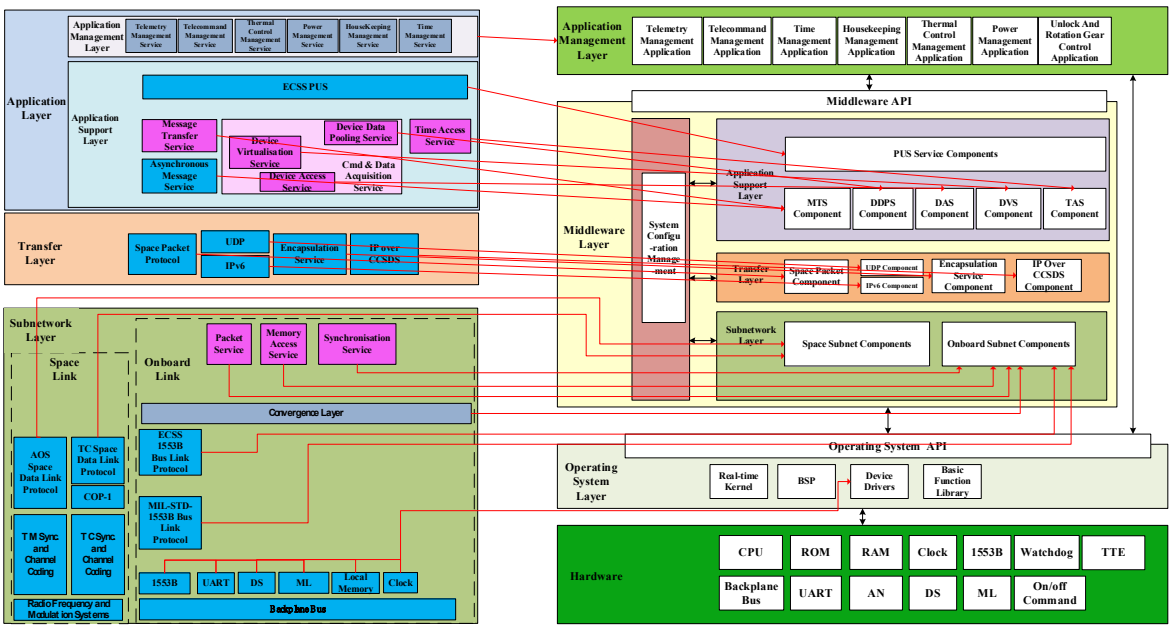

Figure 3-2: Relationship between Service and Protocol Architecture and Software Architecture

3.3 RELATIONSHIP BETWEEN SOIS AND OTHER STANDARDS

3.3.1 GENERAL

As specified in 3.2.3, the service and protocol architecture involved in CAST flight software contains services and protocols from CCSDS SLS, SIS, SOIS, and ECSS PUS. This section will focus on the relationship between SOIS services and other standards, including the

a) relationship between SOIS services and PUS services;

b) relationship between SOIS services and SLS protocols;

c) relationship between SOIS services and SIS protocols.

3.3.2 RELATIONSHIP BETWEEN SOIS SERVICES AND PUS SERVICES

3.3.2.1 Overview

In the service and protocol architecture, 13 services of PUS are adopted, which include telecommand verification service, device command distribution service, housekeeping and diagnostic data reporting service, parameter statistics reporting service, event reporting service, memory management service, function management service, time management service, onboard operations scheduling service, onboard monitoring service, packet forwarding control service, onboard storage and retrieval service, event-action service. SOIS has provided standard service interfaces to upper layer, which can isolate the difference

between data links and protocols. So that SOIS can be used as the underlying supportive service, making the implementation of PUS services focusing more on the algorithm.

PUS services mainly use SOIS Application Support Layer services, the using method is as follows:

a) DDPS is used to acquire data;

b) DVS is used to send command;

c) MTS is used to send and receive packets;

d) TAS is used to get onboard time.

Three recommendations are shown below to describe the relationship between PUS and SOIS services.

3.3.2.2 PUS Onboard Monitoring Service

PUS onboard monitoring service is used to automatically monitor types of onboard specific parameters, and generates an event report if a parameter value is over its threshold. PUS onboard monitoring service needs to acquire the monitored parameter values during its operation, and transfer the generated event report as telemetry packet to the ground or other application processes inside the spacecraft. The acquisition of parameter values can be accomplished by SOIS command and data acquisition service, and the transmission of event report can be accomplished by PUS packet transmission control service and SOIS MTS. This section focuses on the interface relationship between PUS onboard monitoring service and SOIS command and data acquisition service.

SOIS command and data acquisition service consist of DDPS, DVS, and DAS. According to the requirement of PUS onboard monitoring service, DDPS can be used to acquire the monitored parameters.

DDPS provides 11 service primitives. The process of primitive interaction is as follows:

a) onboard monitoring service calls ADD_ACQUISITION_ORDER.request primitive to add order, the parameter Device Value List in the primitive corresponds to the monitored parameters, and the parameter Acquisition Interval in the primitive corresponds to the parameter monitoring interval;

b) DDPS issues an ADD_ACQUISITION_ORDER.indication primitive to return the Acquisition Order Identifier to the onboard monitoring service;

c) onboard monitoring service calls START_ACQUISITIONS.request primitive, using the Acquisition Order Identifier to start the acquisition;

d) DDPS issues START_ACQUISITIONS.indication primitive to pass the result of the request to onboard monitoring service and start the background data acquisition

process, which will use DAS or DVS to acquire the data according to the attribute of the devices in the order;

e) after DDPS accomplishes the acquisition, if the Asynchronous Acquisition Indication Flag is set in the order, it then issues an ACQUISITION.indication primitive to the onboard monitoring service;

f) when the indication is received or the running cycle is due, the onboard monitoring service will use a READ_SAMPLES.request primitive to acquire the data samples;

g) onboard monitoring service issues a READ_SAMPLES.indication primitive to deliver the Samples and Result Metadata to onboard monitoring service;

h) onboard monitoring service judges the parameters and actions according to certain algorithm using the Samples and Result Metadata.

In the process as described above, the cooperation between the services is the key to achieving the parameter conversion between the services. In step a), mapping the Para_id in the PUS Onboard Monitoring Service to the Device Value List in DDPS is a problem to be solved. A design example is given for reference. In this example, the Para_id is correlated with the parameter code of the engineering application. And a special parameter is used for each subsystem. For instance, the parameter code TMSXXX is used to represent the parameters of data management subsystem. Para_id can be converted according to table 3-2.

Table 3-2: Settings of Para_id

Subsystem identification	Corresponding parameter channel ID
5bit (corresponding to TMS)	11bit (corresponding to XXX)

The first 5 bits are used to identify the subsystems. For example, 0x07 is the ID of data management subsystem, and 0~63 are the corresponding parameter channel IDs of the collected analog channels. In this way, the 1st to 64th analog communication channels of the data management subsystem can be represented by TMS001~TMS064. The corresponding Para_id ranges from 0x3800 to 0x3840.

In DDPS, Device Value List consists of an array of identifiers, including the 16-bit Device_id, 16-bit Value_id, and 8-bit Service_type, which represents the use of DAS or DVS. A look-up table for Para_id and Device Value List is created in DDPS. When onboard monitoring service needs to add new monitored parameters, Para_id can be converted to Device_id, Value_id, and Service_type, and then the set of converted parameters can be input into the Device List through ADD_ACQUISITION_ORDER.request primitive. When DDPS is collecting in the background, Device_id and Value_id can be used as input parameters of the underlying DAS or DVS, so that the device parameters acquisition is completed.

3.3.2.3 PUS Onboard Operations Scheduling Service

PUS onboard operations scheduling service is used to provide the time-tagged command sending control. It can receive commands from other ground or spacecraft applications, add or delete commands that need to be regularly executed in its schedule, and download the schedule. This service usually runs periodically. It obtains the spacecraft onboard time through TAS and compares the onboard time with Time Tag in the enabled sub-schedule. The command is sent to the destination by MTS when time is due. This service mainly uses SOIS TAS and MTS.

The application process of PUS onboard operations scheduling service is illustrated through an example of sending a PUS telecommand packet when time is due. In this example, the application process of APID_A in onboard operations scheduling service is set to 0x421, and the destination application process APID_B is set to 0x422.

a) onboard operations scheduling service (application process APID_A) calls Register.request primitive of MTS to complete the registration;

b) telecommand packet destination application process APID_B uses the same process to complete registration and sends an Assert_invitation.request invitation with the subject of command message;

c) application process APID_A replies Assert_invitation.indication to application process APID_B to accept the invitation;

d) onboard operations scheduling service runs periodically, calling the TIME.request primitive of TAS to get onboard time, and TAS issues TIME.indication to return the onboard time to onboard operations scheduling service;

e) onboard operations scheduling service compares the onboard time with the Abs/Rel Time Tag in the enabled sub-schedule and calls the Send.request of MTS to send the telecommand packet to the destination application process APID_B, which is identified by APID of Telecommand packet when time is due;

f) the destination application process APID_B receives the telecommand packet through Message.indication primitive of MTS and performs the further processing.

In step f), the mapping of parameters in the Send.request primitive is a problem to be solved. The complete form of Send.request primitive is Send.request (SAP, continuum ID of destination, unit ID of destination, module number of destination, subject ID, [priority], [flow label], application data length, [application data], [context]), in which the data destination is identified by three parameters, namely *continuum ID of destination*, *unit ID of destination*, and *module number of destination*.

In order to facilitate the transmission of PUS packet, the upper interface of MTS is encapsulated, and an AMS node is identified by APID. The continuum ID, Unit ID, and module number used by APID and AMS are stored in the address mapping table of AMS internal node. For example, after APID_B registration, continuum ID, unit ID, and module number can be queried in the address mapping table through APID_B.

3.3.2.4 PUS Device Command Distribution Service

PUS device command distribution service is used to send device command in real time. It includes three service subtypes. This section takes distributing register load commands sub-service as an example to illustrate its relationship with SOIS services. PUS distributing register load commands sub-service receives requests from ground users or spacecraft applications and sends the register load command. SOIS command and data acquisition services consist of DDPS, DVS, and DAS. According to the demand of distributing register load commands sub-service, the DVS can be used to complete register load commands distribution. The detailed application process is as follows:

a) When distributing register load commands sub-service is activated to run, a Transaction Identifier is allocated to each command in the telecommand packet, and parameters in the command are converted to corresponding parameters in DVS. The converted parameters along with Transaction Identifier are used as the input parameters of the COMMAND_DEVICE.request primitive of DVS, and command is recorded into the queue of executing commands by DVS.

b) After DVS sending the commands through bottom-level DAS, the transferring results return to distributing register load commands sub-service through the *Transaction Identifier* and *Result Metadata* in the COMMAND_DEVICE.indication primitive. In this process, Transaction Identifier comes from step a).

c) Distributing register load commands sub-service finds the corresponding commands in the queue of executing commands according to the Transaction Identifier, and performs further process according to the returned results.

In the process, as described above, the conversion between the register address in distributing register load commands sub-service and DVS primitives is a key issue. The register address is used in the data domain of PUS distributing register load commands sub-service. Similar to the previous telemetry parameter numbering, the instruction code TCSXXX can be used to represent the instructions of the data management subsystem and is associated with the register address. Addresses can be defined using the following rules:

Table 3-3: Settings of Register Address

Subsystem identifier	Corresponding register load commands channel code
5bit	11bit

Subsystem identifier is defined by the project. Taking the data management subsystem as an example, if the data management computer in the system has two register load command channels, the identifier can be designed as table 3-4.

Table 3-4: Example of Register Load Command Channel Identifier

Command ID	register load channel	Subsystem identifier (5bit)	Corresponding register load command channel code (11 bit)
TCS001	ML1	0x7	0
TCS002	ML2	0x7	1

In the computer DVS, one register load command virtual device can be configured, and the corresponding virtual device table is listed as shown in table 3-5.

Table 3-5: Virtual Device Table

Virtual Device ID (16 bit)	The total number of virtual value ID (16 bit)	Address of virtual value resolution table (32 bit)
513 (register load command virtual devices)	2	Virtual value resolution table address of the register load command virtual devices

Virtual value resolution table of the register load command virtual devices is shown as table 3-6.

Table 3-6: Virtual Value Resolution Table of the Register Load Command Virtual Devices

Order number (Virtual value)	Corresponding physical device	Physical device ID (32 bit)	Physical value ID	Length	Offset	Data buffer address	Device access type
0	ML1 command sending device	8	0	1024	0	null	Universal device access DACP
1	ML2 command sending device	9	0	1024	0	null	Universal device access DACP

In the distributing register load commands sub-service, the look-up table to configure a register address along with the virtual device ID and virtual value ID is illustrated as table 3-7.

Table 3-7: Virtual Value Resolution Table of the Register Load Command Virtual Devices

Register Address 16 bit (Corresponding to the command ID)	Virtual device ID 16 bit	Virtual value ID 16 bit
TCS001~TCS002	513 (ML command virtual device)	0~1

In the distributing register load commands sub-service, virtual device ID and virtual value ID are obtained by looking for table 3-7 according to register address and are then transmitted to DVS together with data. DVS can search the Virtual Device Table by virtual device ID to get the corresponding virtual value resolution table address and then look for the virtual device according to the virtual value ID. The value resolution table obtains the corresponding physical device ID, physical value ID, as well as other parameters, and sends the data through DAS.

3.3.3 RELATIONSHIP BETWEEN SOIS SERVICES AND SLS PROTOCOLS

Protocols from SLS domain used in CAST FUHSI include TC, AOS, SPP, and Encapsulation Protocol. Telecommand function can be accomplished by TC and SPP, or Encapsulation Protocol together with ECSS PUS. Telemetry function can be fulfilled by AOS and SPP, or Encapsulation Protocol together with ECSS PUS. SPP or Encapsulation Protocol are the key to building the relationship between SOIS services and SLS protocols.

For telecommand, the application process is as follows:

a) TC space data link protocol receives and processes TC transfer frame; space packets or encapsulation packets will be extracted and delivered to the Transfer Layer using MAPP.indication or VCP.indication primitive, based on the service type the transfer frame used.

b) SPP of the Transfer Layer gets the space packet in order through PACKET.request primitive, together with APID and other information, which will then route the packet according to the APID. Space packets will be delivered to the user through PACKET.indication primitive. Encapsulation Protocol gets packet from the interface which is supplied to underlying protocol, sends the packet to user.

The user mentioned here has two types: one refers to services, protocols, or other applications that are above the Transfer Layer; the other refers to other nodes, for which the Subnetwork Layer Packet Service will be used, and which will be responsible for routing the packet.

For telemetry, the application process is as follows:

a) User requests to transfer a space packet using PACKET.request primitive of SPP, together with parameters such as space packet and APID. Additionally, user transfers an encapsulation packet using Encapsulation Protocol, together with parameters.

b) SPP routes the packet according to APID. When the destination of the APID is the ground, PACKET.request primitive of AOS will be called to transfer the packet to AOS protocol entity. Encapsulation Protocol gets data transmitted to ground, encapsulates the data into a packet, and calls the PACKET.request primitive of AOS to transfer the packet to AOS protocol entity.

c) AOS protocol entity creates the transfer frame and sends it to the ground.

3.3.4 RELATIONSHIP BETWEEN SOIS SERVICES AND SIS PROTOCOLS

AMS and UDP/IP of SIS are used in CAST flight software. The former is the underlying service of MTS, helping accomplish the function of message transfer. Data transmission can be fulfilled by the latter together with Encapsulation Protocol, SOIS protocols, and PUS protocols of ECSS. Additionally, AMS can also be used between two spacecraft or between spacecraft to ground.

In the application process, through analyzing the recommendation to tailor AMS in MTS and considering the complexity and efficiency of software implementation, AMS has been further tailored, which includes:

a) There is no single central node, that is, Configuration Server of AMS in the spacecraft, all registers have equal authority, which forms distributed network architecture that has no central node.

b) MIB maintained by MTS includes a user table and information requirements table. The user table contains all user IDs and addresses in the spacecraft; the information requirements table contains the expected subjects, IDs of information requesters, and priorities of information.

c) Management information is synchronized when MTS starts. Synchronization request will be sent to MTS of other devices. User table and information requirements table will be acquired from other devices, in order to synchronize the local ones.

d) MTS of different devices has no cyclic exchange of Synchronization service, except for initial stage.

3.3.5 RELATIONSHIP BETWEEN SOIS SERVICES AND UDP/IP PROTOCOLS

The UDP/IP protocols are used in CAST FUHSI in the transfer layer, and IPv6 is the key to connect the onboard subnet and space subnet.

The relationship is as follows:

a) The SOIS Application Support Layer services such as MTS can use UDP/IP to send and receive messages.

b) IPv6 protocol use SOIS subnetwork packet service to send and receive IPv6 datagrams.

An example is shown as follows.

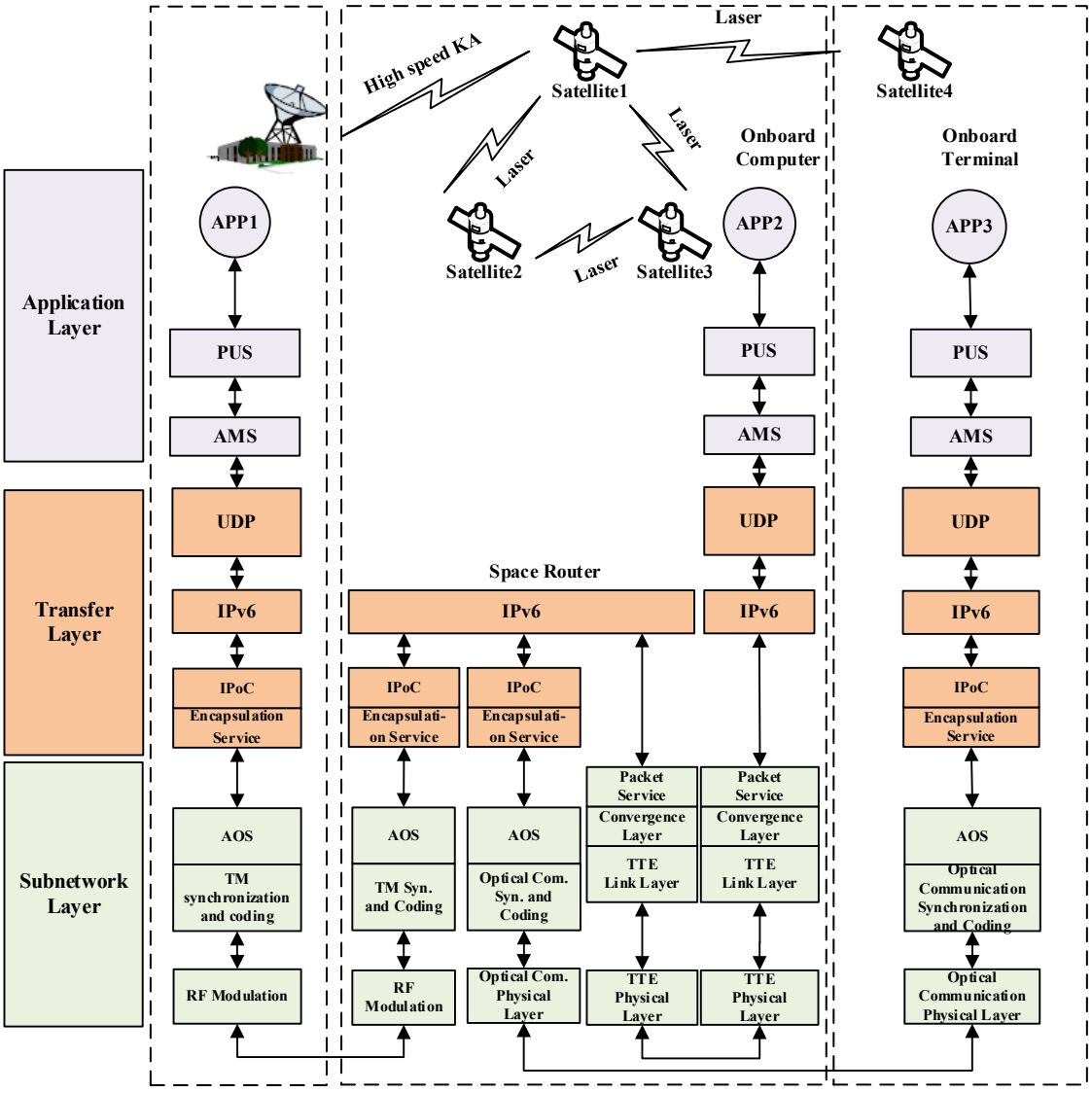

Figure 3-3: Example of Protocol Configuration

4 RELATIONSHIP BETWEEN SOIS SERVICES

4.1 GENERAL

The service and protocol architecture shown in 3.2.3 uses five services from the SOIS Application Support Layer, SPP in the Transfer Layer, and three services from the SOIS Subnetwork Layer. The services and protocols in different layers have relevant naming mechanisms, which have some relationship. How to establish the relationship between the services and protocols of different layers is a key issue in the application of SOIS services and protocols.

The chapter shall include:

a) the naming mechanism;

b) the major service relationship and addressing mechanism, which shall include the relationship between MTS and services below, the relationship between CDAS and services below, the relationship between TAS and services below, and the relationship between services of the Transfer Layer and services of the Subnetwork Layer.

4.2 NAMING MECHANISM

The hierarchy of SOIS services naming is shown in figure 4-1.

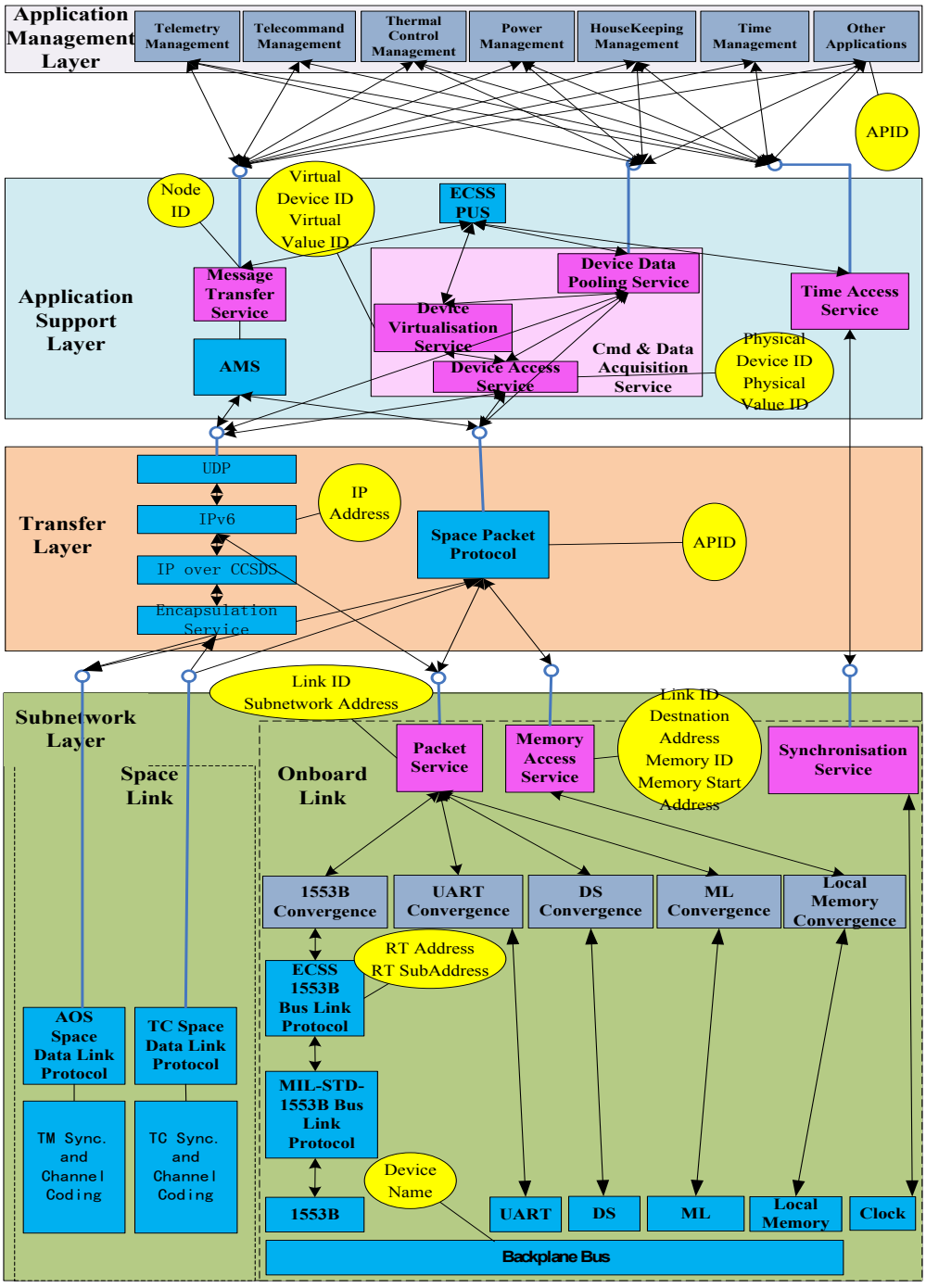

Figure 4-1: The Hierarchy of Naming

a) APID is used to distinguish the applications in the Application Management Layer.

b) In the Application Support Layer, Node ID is used to identify the users of the Message Transfer Service. The main names of DVS include Virtual Device Identifier and Value Identifier. The main names of DAS include Physical Device Identifier and Value Identifier. In actual application, APID or IP Address is used as Node ID directly in MTS. Physical Device Identifier and Virtual Device Identifier are assigned

for each interface of the device module, and the data of the interface can be identified with the Physical Value Identifier.

c) In the Transfer Layer, the packets are routed with APID or IP Address. Each device in the network is assigned one or more APID or IP Address.

d) In the Subnetwork Layer, the Packet Service has several names, such as Link ID, Subnetwork Address, and so on. In application, Link ID together with Subnetwork Address comprises the Packet Destination Service Access Point (PDSAP) address in the Packet Service primitive. Packet Service chooses the convergence link by Link ID. When 1553B link is chosen to implement Packet Service, the 1553B convergence service converts the Subnetwork Address to an RT address and one or more RT sub-addresses. The Memory Access Service has names including Link ID, Subnetwork Address, Memory ID, Start Memory Address, and so on. Link ID, together with the Subnetwork Address, comprises the Destination Address in the Memory Access Service primitive. Because packets transmitted from the Transfer Layer to Subnetwork Layer include the source address and destination address, the Packet Source Service Access Point (PSSAP) address of the Packet Service primitive and the MASAP address of Memory Access primitive are not used. The driver of a device is identified by the device name.

The naming relationship of different layers and specific addressing mechanisms are shown in 4.3.

4.3 MAJOR SERVICES RELATIONSHIP AND ADDRESSING MECHANISM

4.3.1 IDENTIFICATION OF RELATIONSHIPS BETWEEN CDAS AND SERVICES UNDERLYING

CDAS includes DDPS, DAS, and DVS. DDPS gets device data through DAS or DVS. DVS sends commands to devices or acquire data from devices through DAS.

CDAS establishes the relationships with services below through DAS. The major functions of DAS include:

a) identifying the devices and parameters in the device access request of users;

b) selecting the corresponding access service type, calling the access services of lower layer through Transfer Layer, or sending the access requests to DAS on the remote device;

c) receiving access results, storing them for the user, or sending the results to DAS on the remote device;

d) submitting the access results obtained to the user.

The following highlights the relationship between DAS and the services below. In the DAS Recommendation Book, the interaction between DAS and the underlying services includes Packet Service and Memory Access Service.

In CAST FUHSI, two DAP types of DAS are further divided, including:

a) DAP based on Packet Service. The protocol engine in DAS exchanges packets with the protocol engine in the device through the underlying Packet Service. The protocol engine in the device performs the actual operations on the device. This class includes three types:

   1) Packet-send DAP: devices send packets asynchronously. A typical application scenario is that the processor software collects the data of other subsystem devices attached to the DS interface and collects the packet data from other subsystem devices (only support Packet Service) through 1553B bus.

   2) Packet-receive DAP: devices receive packets. A typical application scenario is that the processor software sends ML commands to other subsystem devices attached to the ML interface and transmits packet data to other subsystem bus terminals (no Application Support Layer, only support Packet Service).

   3) DAP based on remote packet access: both devices communicate with each other through remote device access protocol to enable remote device access. A typical application scenario is that the computer gets accesses to the interfaces of other subsystem devices through remote access DAP with 1553B bus.

b) DAP based on Memory Access Service. The protocol engine in DAS determines the location of the memory to be read or written to and gets access through the underlying Memory Access Service. This class includes 2 types:

   1) Universal memory access DAP: the computer performs read and write operations to the memory through Memory Access Service directly. A typical application scenario is that the processor module acquires the internal state telemetry of other modules.

   2) Analog data access DAP: the computer needs to filter the data acquired through the universal memory access DAP and submits to the user. A typical application scenario is that the processor module collects the analog data of the analog acquisition module.

DAS communicates with Packet Service and Memory Access Service through Transfer Layer uniformly. The following illustrates the process with commands sending program:

a) The user calls COMMAND_DEVICE.request of DAS to send a command to a device. The incoming parameters include Physical Device Identifier, Value Identifier, data, etc.

b) According to the Physical Device Identifier, DAS determines that the device can be communicated with through Packet Service. Then it can obtain the corresponding device APID based on Physical Device Identifier and Value Identifier, and transmits information such as APID and data to the Transfer Layer with the PACKET.request primitive.

c) Transfer Layer routes packets according to APID and sends data through the Subnetwork Layer Packet Service. The specific process of the Transfer Layer is described in 4.3.4.

If the command is issued by a remote device in the above procedure, the DAS in step b) knows that the device is a remote device, organizes the command of DAS as one or more space packets, and transmits the packets to the remote device through Transfer Layer. The remote DAS receives the command packets through Transfer Layer, resolves the command, and executes the command locally. The results of execution are sent to the initiator's DAS through the Transfer Layer, and the initiator's DAS returns the results to the user.

4.3.2 IDENTIFICATION OF RELATIONSHIPS BETWEEN MESSAGE TRANSFER SERVICE AND SERVICES UNDERLYING

The PDU generated by MTS needs to be transmitted through the lower layer service. In the AMS standard (reference [13]), the lower transfer services can use TCP, UDP, FIFO, vxmq, smmq, and other protocols or mechanisms to implement data transmission. In CAST FUHSI, the data shall be transmitted through the Transfer Layer in a unified way, which currently supports SPP and UDP/IP and can be further extended.

Taking the message transmission of MTS and SPP as an example, its interaction process with the underlying services is as follows:

a) After registration and invitation by the receiver, the user sends a Send.request to MTS, and the destination is identified by APID.

b) MTS organizes the data into a PDU and checks whether the application procedure of the destination is local according to the APID lookup table. If the destination is local, the destination application procedure is sent through the local memory directly. If it is not local, PDU, destination APID, and other parameters are sent to the Transfer Layer together.

c) According to APID, the Transfer Layer can get subnetwork Packet Service parameters and send data to the Packet Service, which sends the data to the destination with the convergence link between processors.

d) The destination receives the data through the Subnetwork Layer, and passes the data to the Transfer Layer. The Transfer Layer sends the data to MTS, which submits the data to the user.

4.3.3 IDENTIFICATION OF RELATIONSHIPS BETWEEN TIME ACCESS SERVICE AND SERVICES UNDERLYING

TAS interacts with the Synchronization Service of the Subnetwork Layer and is used to acquire the spacecraft time. In the specific application process, time is divided into two types, that is, absolute time and relative time. Taking the absolute time acquisition as an example, the interaction process is as follows:

a) the user invokes TIME.request primitive of TAS;

b) TAS invokes the TIME.request primitive of the Synchronization Service upon receipt of the request;

c) the TIME.request primitive of the Synchronization Service invokes the corresponding device driver of the clock to obtain the current spacecraft time and returns the time to TAS through TIME.indication;

d) TAS receives the time and returns it to the user through its TIME.indication primitive.

In addition to acquiring the spacecraft time, TAS also provides the ALARM and METRONOME functions, both of which are supported through the timer of the operating system.

4.3.4 IDENTIFICATION OF RELATIONSHIPS BETWEEN TRANSFER LAYER AND SUBNETWORK LAYER

The Transfer Layer can interact with the Packet Service and Memory Access Service of the Subnetwork Layer.

Taking sending data to the ML interface as an example, the interaction process between the Transfer Layer and the Packet Service is as follows:

a) the upper layer service or protocol invokes the PACKET.request primitive of the Transfer Layer to send data;

b) the Transfer Layer routes packets according to APID in the primitive, obtains the corresponding Link ID and Subnetwork Address (corresponding to PDSAP address parameter of the Subnetwork Layer Packet Service), service type, channel, priority and so on, and invokes PACKET_SEND.request primitive of Subnetwork Layer Packet Service;

c) the Packet Service of the Subnetwork Layer gets the corresponding link convergence from the lookup table based on the link ID and calls the sending interface of ML link convergence;

d) ML Link Convergence gets the device name of device driver, and sends the data through the driver according to the pre-configured device driver parameters.

Taking analog data acquisition as an example, the interactive process between Transfer Layer and Memory Access Service is as follows:

a) The upper layer service (e.g., DAS) organizes the read commands of the Memory Access Service into packets, which contain all parameters of the command, and then invokes PACKET.request primitive of Transfer Layer to send data.

b) The Transfer Layer routes packets according to APID in the primitive, knows the corresponding service is Memory Access Service, and sends data to Memory Access Service.

c) The Memory Access Service resolves the Memory ID from the packet, gets the corresponding device driver name from lookup table, and calls the device driver to read the device data. After returning the device data, the result is organized into a response packet, and the destination is DAS. The packet and the destination APID are forwarded to the Transfer Layer.

d) The Transfer Layer forwards the response packets to DAS according to the destination APID.

5 RELATIONSHIP BETWEEN SOIS SERVICES AND DEVICE HARDWARE

5.1 GENERAL

It is also very important to build the relationships between SOIS services and the specific devices of the CAST avionics system while applying the SOIS services to CAST FUHSI. The hardware-related services in SOIS mainly include:

a) Packet Service in subnetwork layer and convergence layer functions, which need to be provided by specific onboard links so there are mapping relationships between links (e.g., 1553B bus link, DS/ML interface) and these functions.

b) Memory Access Service, which offers memory read/write operations. It should establish a relationship to these memory operations of actual hardware within the architecture.

c) Synchronization Service in subnetwork, which is a time-related service; therefore it should deal with the clock interface of hardware.

d) DAS, DVS, and DDPS, which are related to the hardware devices in the system and need to establish the mapping relationship with each device in the spacecraft.

This section first analyzes the hardware types in CAST avionics system, and then gives specific access methods in accordance with the classification of hardware.

5.2 DEVICE TYPES ANALYSIS IN CAST AVIONICS SYSTEM

From the intelligence level point of view, devices in spacecraft can be divided into three categories:

a) Intelligent nodes: these nodes provide strong processing ability and support a complete protocol stack with message processing capabilities, which can handle peer-to-peer communication. The protocols (e.g., MTS) used in these nodes can perform the functions including: subscribing a set of interested data without knowing the senders, publishing their own data without knowing the receivers, querying interested data, etc. Typical representatives of intelligent nodes are common processor modules of OBDH onboard computers, attitude control computer, and payload management computer.

b) Simple intelligent nodes: these nodes are slightly less intelligent than intelligent nodes and only support transfer and subnetwork services, while having space packet processing ability. Typical representatives of simple intelligent nodes are the telemetry data collecting module and command send module.

c) Non-intelligent nodes: these nodes are typically controlled by intelligent nodes or simple intelligent nodes, which can send/receive original data or space packets.

Typical representatives of non-intelligent nodes are devices which are attached to On/Off command, AN, DS, ML interface, and so on.

d) The following parts take a design of spacecraft avionics system, for example, and demonstrate how its nodes work. The instance of avionics system includes one Spacecraft Management Unit (SMU) and one Spacecraft Data Interface Unit (SDIU). SMU and SDIU are assembled by standard modules. Modules are connected via backplane bus. In this example, 1553B bus is used for communication between SMU, SDIU, and other subsystem devices. The composition diagram is shown in figure 5-1.

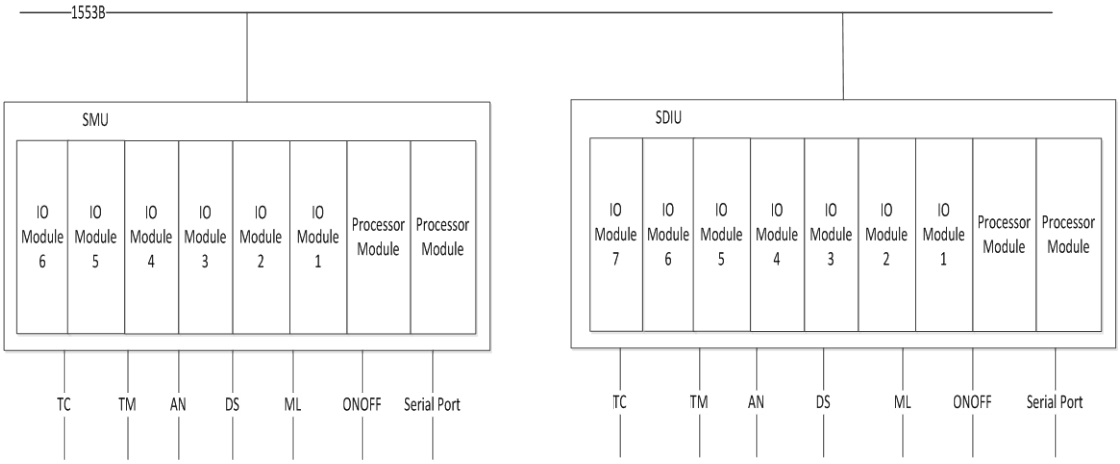

Figure 5-1: Hardware Platform Composition Diagram of Avionics System

The designed avionics system provides external interfaces including:

a) TM/TC interface;

b) command interface: including ON/OFF interface, ML interface, etc.;

c) data collection interface: including analog collection interface, DS interface, etc.;

d) bus interface: including 1553B bus, etc.

SMU and SDIU both have strong processing abilities and processor modules that can be considered as intelligent nodes. Other subsystem devices that are connected to the 1553B bus can be divided into intelligent and simple intelligent nodes. Intelligent nodes exchange data through MTS, while simple intelligent nodes use the Subnetwork Layer Packet Service for communication.

Other subsystem devices that are connected to the command interface or data collection interface can be considered as non-intelligent nodes. In this case, the processor module in SMU or SDIU can require/distribute data via DAS, PS, MAS, and device drivers.

Subsection 5.3 gives access methods for specific nodes.

5.3 HARDWARE NODES ACCESS METHODS IN AVIONICS SYSTEM

5.3.1 ACCESS METHODS OF INTELLIGENT NODES

The protocol configuration of two intelligent nodes that are communicated through a 1553B bus is shown in figure 5-2.

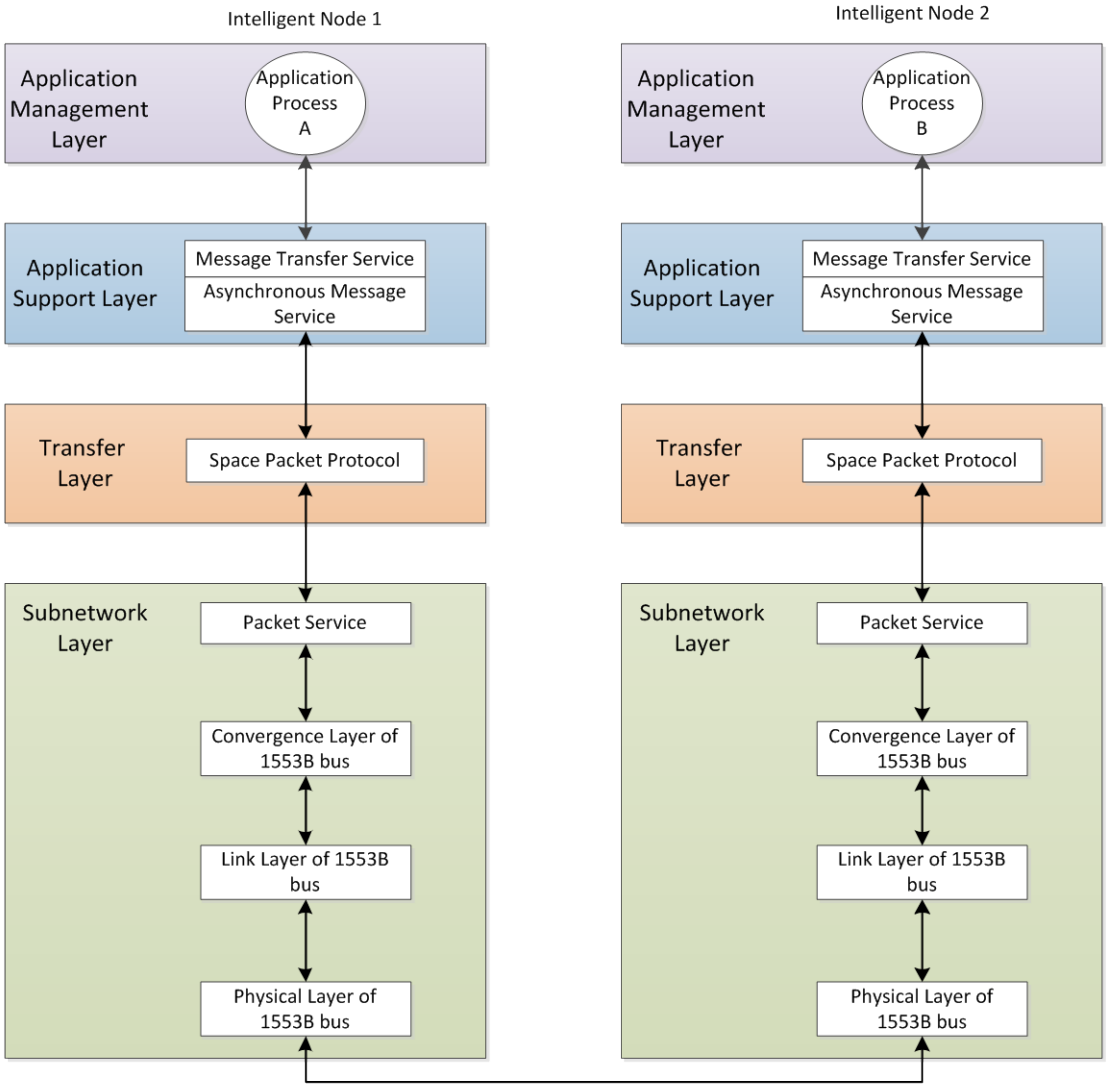

Figure 5-2: Protocol Configuration of Intelligent Nodes through 1553B

Protocols of each layer are configured as follows:

a) Application Layer: the application process in the two intelligent nodes can subscribe, publish, and send messages via the primitives provided by MTS; different nodes can be distinguished by APID.

b) Application Support Layer: MTS uses AMS for implementation; the underlying protocol uses SPP.

c) Transfer Layer: SPP provides packet transmission to the upper layer and uses the Packet Service to send/receive data on the bus or the other links.

d) Subnetwork Layer: Data are sent or received through the Packet Service, 1553B convergence layer, ECSS1553B bus link protocol, MIL-1553B bus link protocol, and physical hardware.

The key to establishing a connection with the hardware is the Subnetwork Layer. Because different data links have different protocols, in order to provide a unified interface to the upper layer, the Packet Service in Subnetwork Layer should provide a uniform packet sending interface to the upper layer applications so as to shield the differences between underlying data links. Then the upper layer applications will not need to be concerned with the differences between different heterogeneous physical link characteristics, interface features, and transmission performance. Once the destination address and the QoS requirements are determined, the upper layer (e.g., Transport Layer) will choose the suitable links according to destination device conditions and data transmission requirements. Finally, the data will be sent to the destination or waypoint through the convergence layer. If one hop cannot reach the destination directly, the data may pass through several waypoints.

In order to achieve the purpose of shielding the underlying data links in the Subnetwork Layer, the convergence layer is a key point. Because different links adopt different protocols, it is difficult to define a unified protocol for convergence layer. In actual implementation, different links may have different convergence protocols. The Packet Service in the Subnetwork Layer selects the convergence layer send interface according to the identification passed from the upper layer (included in the primitive parameters of PDSAP), and the device driver will send the data through actual onboard links.

Taking the 1553B bus link for example, 1553B interface service protocol defined by ECSS can be applied, and a convergence layer can be added on top. The purpose is to add the segmentation function to support up to 64K bytes packets transmitted through the 1553B bus. And in order to match the max packet length between space link and onboard link, TC and AOS protocol also need to support 64K bytes packets. In actual implementation, the convergence layer divides the data into the MTU length of ECSS 1553B protocol support (usually 4K bytes), and provides the corresponding data ID, segment number, and other information, then sends it through the device driver. When the segment data arrives, the convergence layer in the receiver end will read the data through the device driver and assemble each segment correspondingly, then commit to the upper layer application once the whole packet is received.

The protocol configuration of two intelligent nodes that are communicated through TTE is shown in figure 5-3.

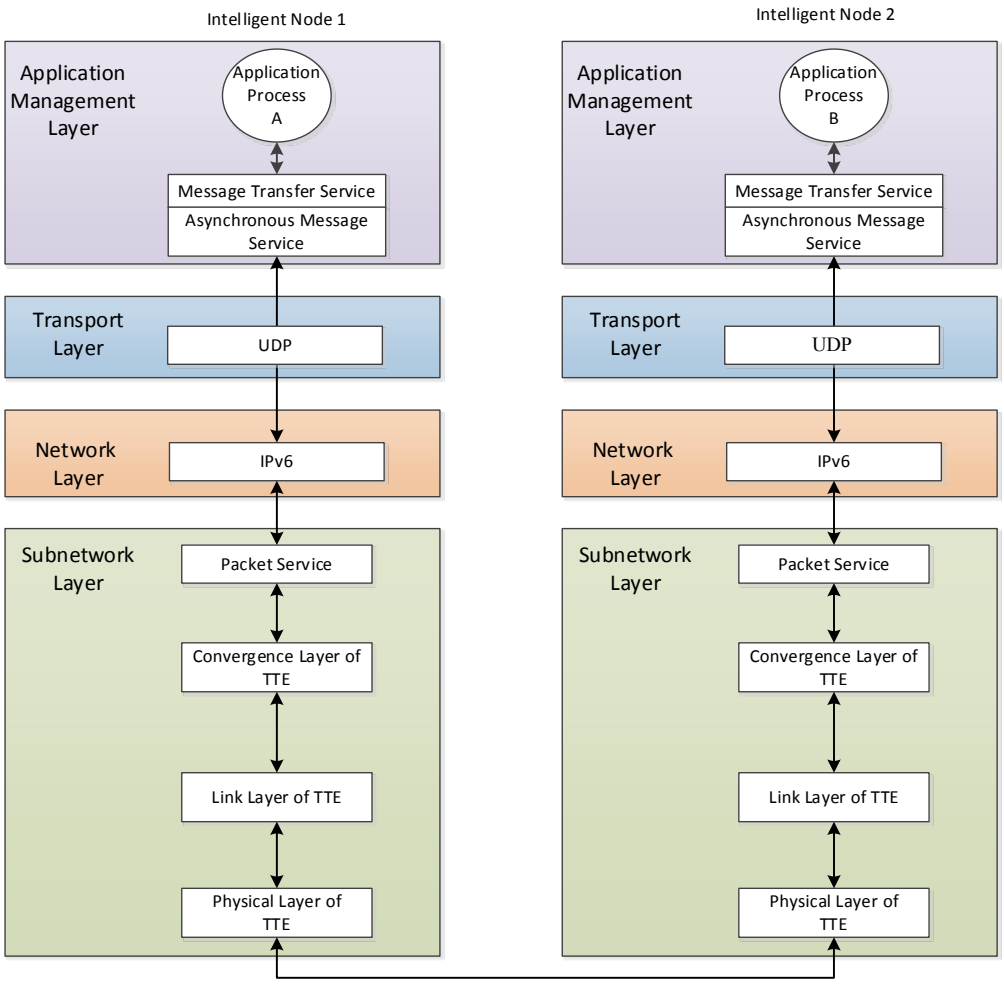

Figure 5-3: Protocol Configuration of Intelligent Nodes through TTE

Protocols of each layer are configured as follows:

a) Application Management Layer: the application process in the two intelligent nodes can subscribe, publish, and send messages via the primitives provided by MTS;

b) Transport Layer: UDP provides packet transmission to the upper layer and uses the IPv6 as the underlying layer protocol to transmit data;

c) Network Layer: IPv6 provides packet transmission to the upper layer and uses the Packet Service as the underlying layer service to transmit data on TTE bus;

d) Subnetwork Layer: data are sent or received through the Packet Service, TTE convergence layer, TTE bus link protocol, and physical hardware.

The key to establishing a link layer connection with the high-speed Ethernet is the TTE Protocol of Subnetwork Layer. Additionally, in order to network with other nodes, the IPv6 and UDP protocol is the key.

5.3.2 ACCESS METHODS OF SIMPLE INTELLIGENT NODES

The protocol configuration of an intelligent node communicating with a simple intelligent node through 1553B bus is shown in figure 5-4.

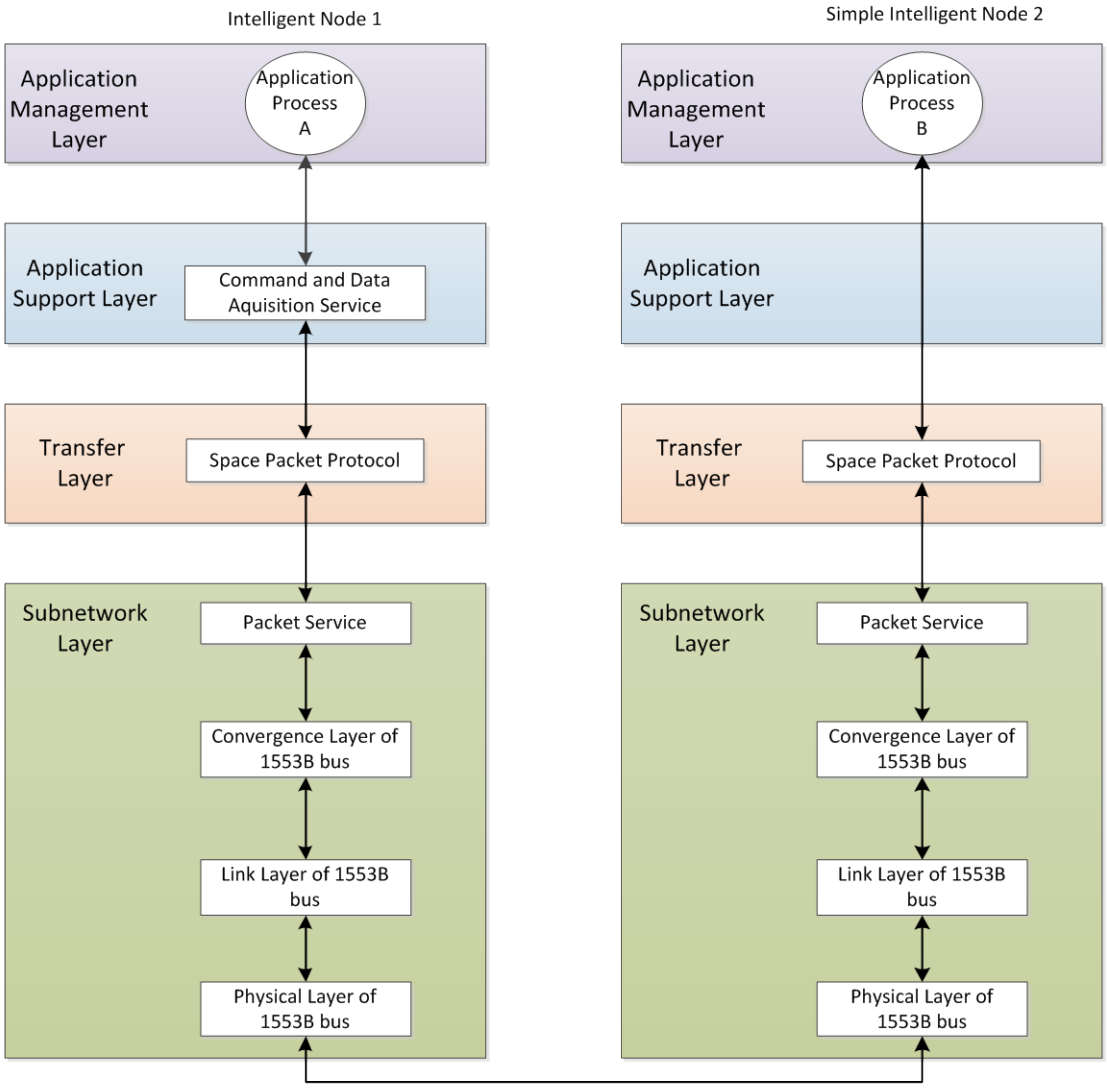

Figure 5-4: Protocol Configuration of Accessing Simple Intelligent Nodes

Protocols of each layer are configured as follows:

a) Application Layer: the application process in intelligent nodes can access data from simple intelligent nodes through CDAS.

b) Application Support Layer: CDAS uses SPP, which is provided by the underlying layer. For example, DAS of CDAS can be used to send/receive a space packet to simple intelligent nodes.

c) Transfer Layer: SPP provides packet transmission to the upper layer and uses the Subnetwork Layer Packet Service to send/receive data on the bus or the other links.

d) Subnetwork Layer: data are sent or received through the Packet Service, 1553B convergence protocol, ECSS1553B bus link protocol, MIL-1553B bus link protocol, and physical hardware.

The design of DAS in CDAS is the key in the process mentioned above. A list of simple intelligent nodes should be built in DAS that is implemented in intelligent nodes, with the different kinds of DAPs configured for each node in this list. For the above example, The DAP for simple intelligent nodes is a packet-based DAP with three types and can be configured according to device implementation and connection mode. When a simple intelligent node wants to send a packet to an intelligent node asynchronously, its DAP can be configured as a packet-send DAP, as mentioned in 4.3.1. And when a simple intelligent node needs to receive data from intelligent node, its DAP should be configured as packet-receive DAP, as mentioned in 4.3.1.

5.3.3 ACCESS METHODS OF NON-INTELLIGENT NODES

The protocol configuration of a non-intelligent node exchanging data with an intelligent node using DS, ML, or a serial port interface is shown in figure 5-5.

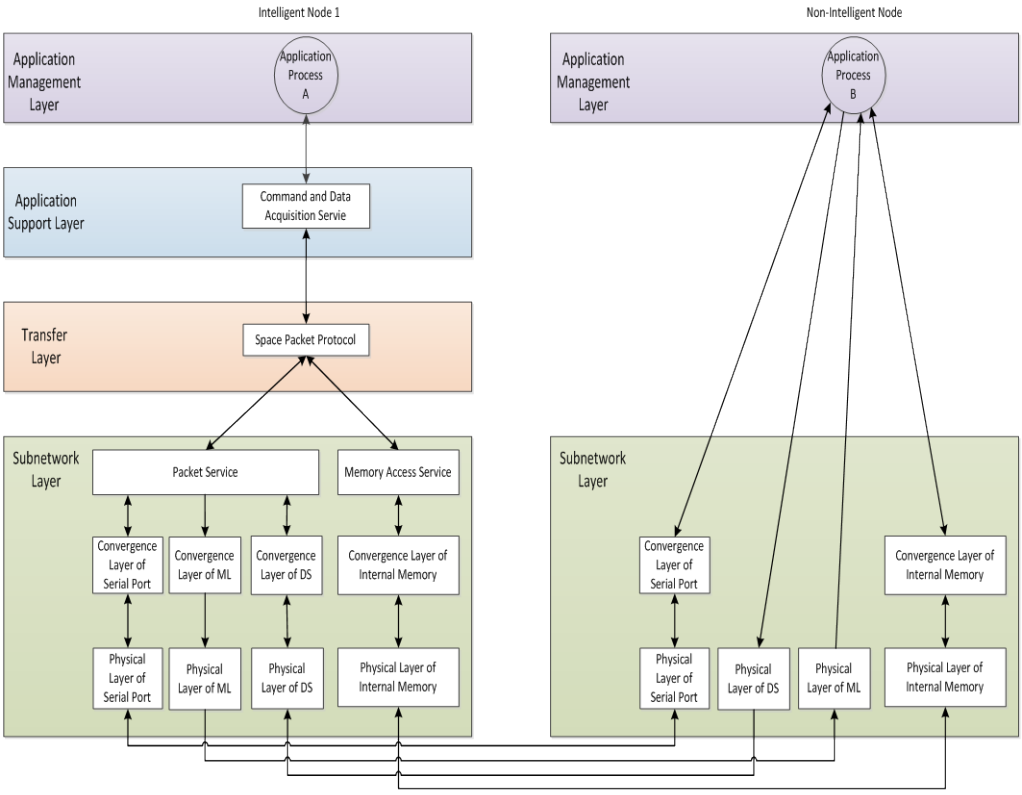

Figure 5-5: Protocol Configuration of Accessing Non-Intelligent Nodes

Protocols of each layer are configured as follows:

a) Application Layer: the application process in intelligent nodes can access data from non-intelligent nodes through CDAS;

b) Application Support Layer: CDAS uses SPP that is provided by underlying service, thus, for example, DAS can send/receive a space packet;

c) Transfer Layer: SPP provides packet transmission to the upper layer and uses the Packet Service to send/receive data on the bus or the other links;

d) Subnetwork Layer: data are sent or received through the convergence protocol of DS, ML, etc. and physical hardware.

For each interface, it is important to configure the appropriate device driver. Convergence protocol will connect the Packet Service, Memory Access Service and the associated device drivers.

If the Packet Service is used in underlying layer, Device and Value Identifier Resolution Table in DAS, routing table in SPP, link selection table in Packet Service and device name configuration table in convergence layer shall all be configured.

Memory Access Service is used to access the interface such as analog collection and command output. In CAST FUHSI, the implementation of Memory Access Service can be divided into the following categories according to physical connection in hardware:

a) Remote Access: through the bus (e.g., 1553B bus) or space link access;

b) Inter-Module Access: through the I/O backplane bus;

c) Intra-Module Access: through CPU bus or local bus (e.g., CPCI) access.

Remote Access is implemented via Transfer Layer configuration, the application in Application Support Layer encapsulates memory access requests as packets, and deploys the destination address (e.g., APID) of the application which handles Remote Memory Access Service, then passes the packet to the Transport Layer. The Transport Layer routes the packet to the target application. The target application receives the memory access request and performs an intra-module or inter-module access operation, then encapsulates the result into a packet and transmits to the source.

Inter-module access and inter-module access are all compatible with specific device drivers. The same device driver can handle multiple devices.

6 APPLICATION OF SEDS

6.1 GENERAL

In CAST FUHSI, the SEDS is used to describe the parameters of the device, the configuration parameters of the system, and the configuration parameters of the service, and the original conceptual communication management in the architecture is transformed into an entity that describes the service configuration and connection relationship of each layer through the SEDS.

The ultimate goal of using SEDS in CAST FUHSI is to automatically generate part of the code through the tool after describing the above data. For the existing components, SESD are mainly used to generate the configuration code. For the new component, SEDS are used to generate the component code and the configuration code. SEDS can also be used as the input for subsequent software testing.

6.2 AN APPLICATION EXAMPLE

In CAST FUHSI, there are 27 software components. A typical example of sending an ML command, which runs through the various layers of the architecture, was chosen, as shown in figure 6-1. The application of SEDS is illustrated by this example. Sending an ML command involves the following services:

a) the Device Access Service of the Application Support Layer;

b) the Space Packet Protocol of the transfer layer;

c) the Packet Service of the subnetwork layer;

d) the ML Convergence Service (Convergence_ML) of the convergence layer.

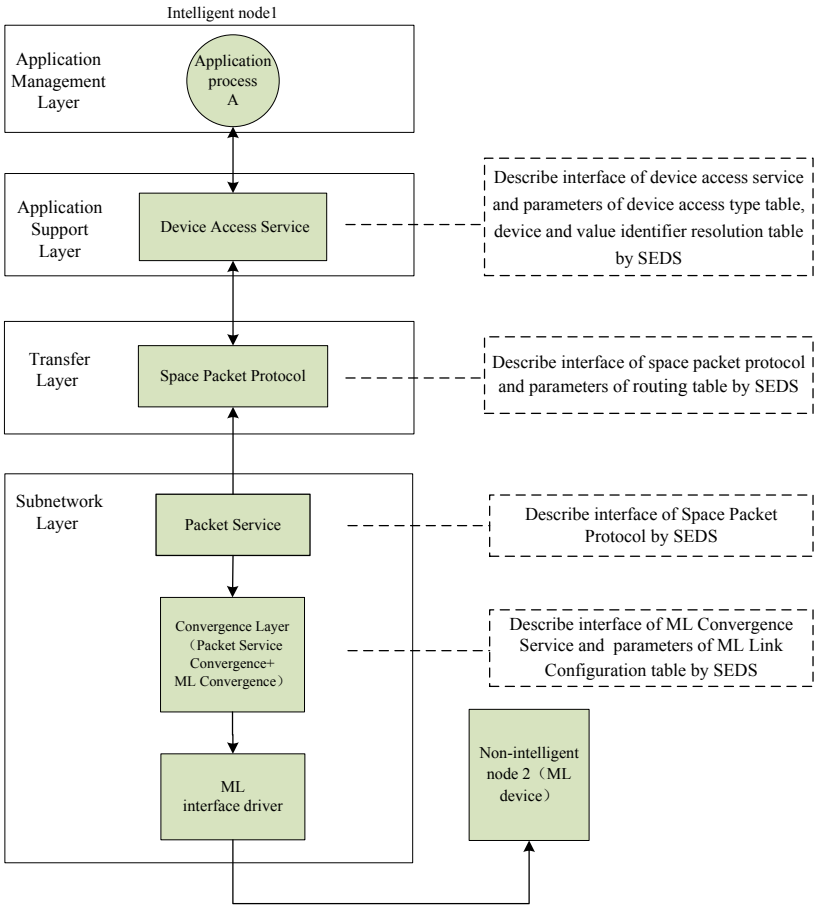

Figure 6-1: Sending a ML Command

6.2.1 INSTRUCTION SENDING PROCESS

The ML command is sent from the Application Layer to the Data Link Layer. The specific sending process is as follows:

a) Application Management layer: the Device Access Service of the intelligent node accesses the simple intelligent node;

b) Application Support Layer: configures the device identifier and value identifier of the non-intelligent node 3: device id = 0x8, value id = 0x0. In the device access service, the device access type, which is sending data to the device DAP, is obtained by searching the device access type table (table 6-1) by device id. Then, through searching the device and value identifier resolution table (table 6-2) by device id and value id, the network address (APID) is found, and then the command and data are encapsulated into a space packet and sent to the space packet protocol of the transfer layer.

Table 6-1: Device Access Type Table

Name	Device id (2Byte)	Corresponding device access type DAP (2Byte)
ML interface 1	0x8	Sending data to the device DAP
ML interface 2	0x9	Sending data to the device DAP

Table 6-2: Device and Value Identifier Resolution Table

Device id (2Byte)	Value id (2Byte)	Network address (2Byte)	Start address (4Byte)	Length (2Byte)
0x8	0x0	0x7 (DEVICE_ID_DEV_3)	0	1000

c) Transfer layer: In the space packet protocol, the routing table is searched by the network address (APID=0x7) , the underlying service is identified as the subnetwork packet service, and the subnetwork identifier is LINK_ML (subnetwork id). The packet is then sent to the subnetwork packet service.

Table 6-3: Routing Table

Network address (2Byte)	Mask (2Byte)	Next hop subnetwork id (2Byte)	Next hop subnetwork address (2Byte)	Assistant parameters (4Byte)
APID_OBC_A(0x420)	0x7E0	LINK_LOCAL(0x0)	0	0
DEVICE_ID_DEV_3(0x8)	0x7FF	LINK_ML1(0x6)	0	0

d) Subnetwork layer: In the packet service, according to the subnetwork id, the link type and the corresponding component instance are found, the externally provided interface is called according to the link type and component instance, and the command is issued.

Table 6-4: ML Link Configuration Information

Link	Link id (2Byte)	Link type (2Byte)	Driver Master (4Byte)	Driver Slave (4Byte)
LINK_ML1	0x6	0	3	1
LINK_ML2	0x7	0	3	2

6.2.2 PARAMETERS AND INTERFACES TO BE DESCRIBED BY SEDS

During the instruction send process, the parameter configuration and interface are as follows:

a) Parameter configuration

The parameter configurations that need to be described are shown in table 6-1, table 6-2, table 6-3, and table 6-4.

See annex C for the description of the parameters by SEDS.

b) Interface

1) Device Access Service of the Application Support Layer

– required interface:

```
status_t (*tpPacketSend_funcp)(uint16_t src_apid, uint16_t dest_apid,
        uint8_t* packet_buffer_p, uint32_t length, uint32_t qos)
```

2) Space Packet Protocol of the transfer layer

– provided interface:

```
status_t tpPacketSend (uint16_t src_apid, uint16_t dest_apid,
        uint8_t* packet_buffer_p, uint32_t length, uint32_t qos)
```

– required interface:

```
status_t (*snPsSend_funcp) (uint8_t qos, uint8_t priority, uint8_t
        channel, uint8_t next_link_id, uint8_t next_sn_address,
        uint8_t *packet_buffer_p, uint32_t length)
```

3) Packet Service of the subnetwork layer

– provided interface:

```
status_t snPsSend (uint8_t qos, uint8_t priority, uint8_t channel,
        uint8_t next_link_id, uint8_t next_sn_address,
        uint8_t *packet_buffer_p, uint32_t length)
```

– required interface:

```
status_t (*snDclMLInterface_funcp)(dcl_ml_com_t *obj_p, uint8_t priority,
        uint32_t length, uint8_t *packet_buffer_p)
```

4) ML Convergence Service of the convergence layer

   – provided interface:

     status_t snDclMLInterface(dcl_ml_com_t *obj_p, uint8_t prority, uint32_t length, uint8_t *packet_buffer_p)

(See annex D for interface descriptions by SEDS.)

7 BENEFITS OF USING STANDARDIZED PROTOCOLS AND SERVICES IN CAST SOFTWARE

7.1 BRIEF INTRODUCTION OF IMPLEMENTATION AND EXPERIMENTATION

In order to validate the software architecture, CAST has implemented all the software components in the architecture. Software components and device drivers have been designed and developed. Based on the hardware platform of avionics system requirements, these components were assembled and tested. An example of CCSDS standard primitive implementation procedures and methods is given in annex A.

Case 1:

The software of hardware platform SDIU and SMU are assembled by software components, which are completely the same, with the runtime parameters and process configured according to the device identification. Task migration and system reconfiguration can be achieved in a machine failure. The total code size of the prototype software is above 50000 lines, and the code size of all software components is above 40000 lines, accounting for 80% of the total code lines.

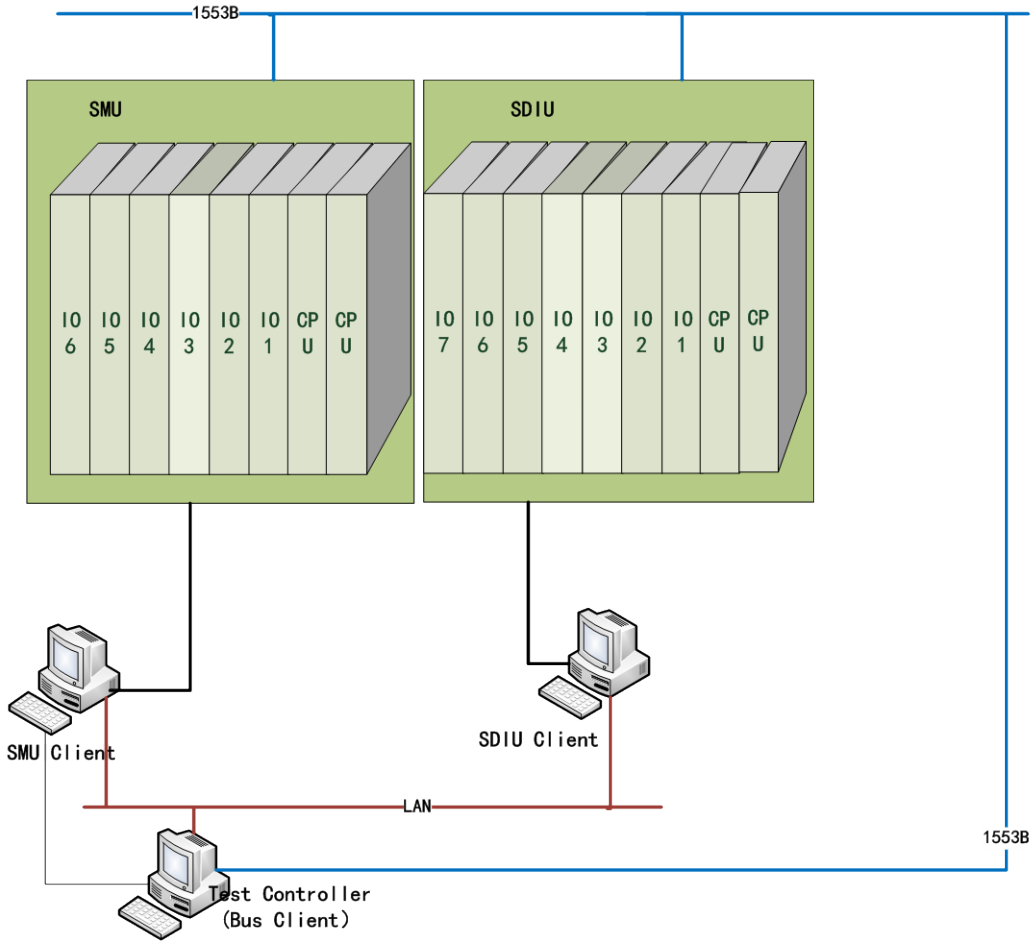

Figure 7-1: Avionics System Hardware Platform and Test System 1

A number of test cases show that CAST FUHSI based on CCSDS standard can not only provide richer, more practical, and more powerful functions than traditional spacecraft software system, but also changes the whole software development mode, improving efficiency and reliability of software development.

Case 2:

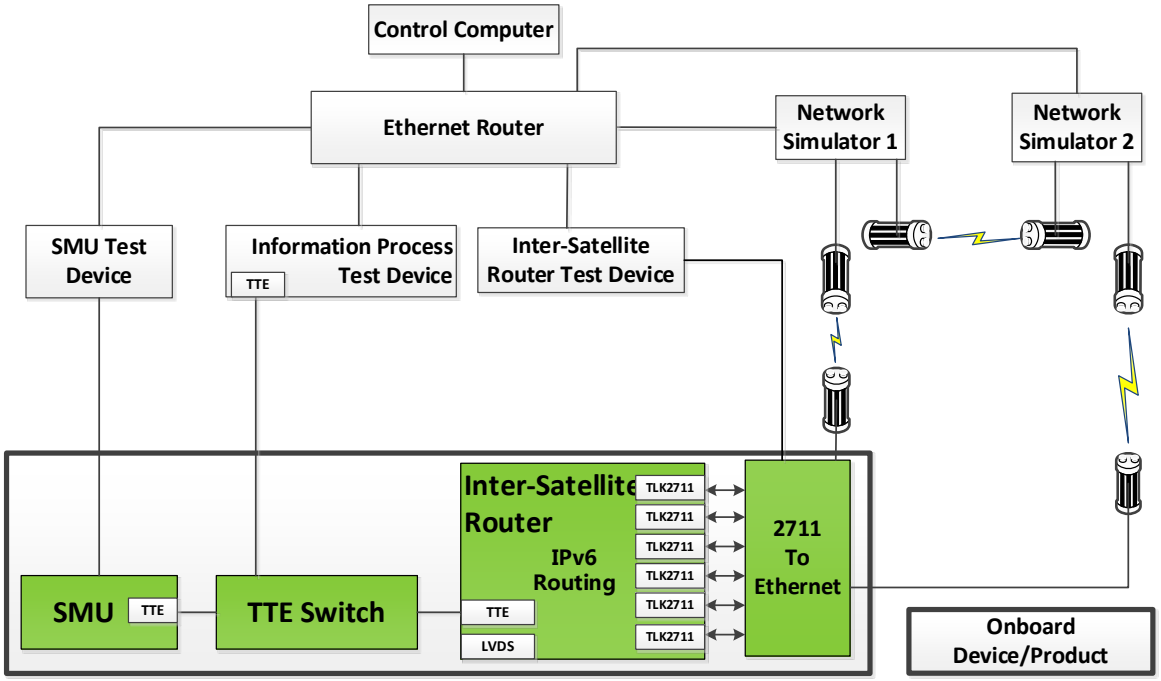

Figure 7-2: Avionics System Hardware Platform and Test System 2

The System is composed of ground equipment and onboard equipment.

Ground equipment includes the Control Computer, Ethernet Router, SMU testing equipment (corresponding to TT&C), Information Process testing equipment (corresponding to camera), Inter-Satellite Router testing equipment (corresponding to Data Transmission Station), and Network Simulator 1 (corresponding to Satellite) and 2 (corresponding to Satellite).

Onboard equipment includes SMU, TTE Switch, and Inter-Satellite Router, which is equipped with TLK2711 interfaces. SMU is responsible for receiving TC frames and sending AOS frames. TTE Switch is responsible for switching MAC frames quickly. Inter-Satellite Router is responsible for connecting networks of intra-satellite and inter-satellite and forwarding data between networks.

A number of test cases show that CAST FUHSI based on the CCSDS standard can not only provide richer, more practical, and more powerful functions than traditional spacecraft software system, but also changes the whole software development mode, improving efficiency and reliability of software development.

7.2 SYSTEM FUNCTION ENHANCEMENT

System function enhancement mainly manifests in the following aspects:

a) The data transmission mechanism is more flexible.

   Traditional spacecraft use the serial data interface to transmit data. The transmission time interval, data length, and transmission destination are all fixed and difficult to change.

   Supported by the CCSDS SPP, SOIS Subnetwork Layer Packet Service, and convergence layer in CAST FUHSI, the user of the system using the serial data interface can control the transmission time interval, the data length of transmission, and the destination on demand. Platform and payload devices can be accessed via any serial data interface, either in raw data or in space packet format. When the data is in raw data format, the system can configure the original data packet processing and routing in advance. If the data is in a standard space packet format, the user can choose the appropriate length, as well as different destinations, and then the system can automatically identify the destination of the data and route data to its destination according to the routing strategy, for example, routing through other onboard equipment, other spacecraft, or ground assets. This mechanism can greatly improve the flexibility and scalability of the system.

b) The software architecture supports the interface replacement without modifying upper-layer application software.

   In some traditional spacecraft, the modification of the onboard software and parameters cannot be avoided when the interface, through which the data is transferred, will be changed.

   In CAST FUHSI, with the CCSDS SPP, SOIS Subnetwork Layer Packet Services and convergence layer, the user can access the system through different interfaces. The Transfer Layer service can automatically transfer data to the right destination according to the user's destination. Even if the user changes the access interface, for example, changing the serial data interface to 1553B bus interface, UART, or another interface, it just needs to set the destination, and then the system can automatically route according to the destination. This mechanism is equivalent to the plug-and-play in initial stage. The device automatic identification mechanism, which will be implemented and added to the architecture in the next step, can further enhance the plug-and-play ability of system.

c) The software architecture supports the system computing capability to expand on demand.

   In CAST FUHSI, the CCSDS MTS, DAS, DVS, DDPS are applied and cooperate with the underlying SPP, the Subnetwork Layer Packet Service, and the convergence layer. The number of processors can be flexibly expanded with the support from underlying hardware. The system can increase the number of processor modules to achieve the task migration and distributed computing, thus enhancing the overall

computing capability of the system. In this process, the data transmission between different processors is completed by the MTS, and different services are adopted according to the actual positions of the communication parties.

a) If the processes are in the same processor module, the MTS uses a local buffer to complete the exchange of messages;

b) If the processes are distributed on two processors, the MTS completes the data transmission through the Transfer Layer, the Subnetwork Layer Packet Service, and the backplane bus convergence layer;

c) When the communication is between two devices connected by the 1553B bus, the MTS performs data transmission via the Transfer Layer, the Subnetwork Layer Packet Service, and the 1553B bus convergence layer.

The difference of underlying links can be completely shielded to the Application Layer, so as to facilitate the development of applications independent of the underlying interfaces.

d) Standard uplink and downlink transmission of large data blocks are supported.

The CCSDS TC protocol, AOS protocol, COP-1 protocol, SPP, Subnetwork Layer Packet Service, and convergence layer are applied in CAST FUHSI, which provides a standardized transmission mechanism for ground users. The user can put a one-time injection of data consisting of maximum 64K bytes packets by TC protocol for automatic segmentation, and in accordance with the COP-1 protocol, to transmit the frames and confirm the results automatically. This procedure can provide a friendly interface to users and can greatly enhance efficiency. Two disadvantages can then be avoided: (a) the need to segment large data packets into multiple small packets and calculate the address one by one, troublesomely; (b) the low transmission efficiency caused by waiting for the confirmation of the previous frame before the user sending the next frame.

e) Space and onboard communications are integrated via transfer layer.

The use of transfer layer, space, and onboard communications can be integrated through network protocols such as SPP, UDP, and IPv6. The data sent by ground to a device onboard a spacecraft via different spacecraft can be routed via the transfer layer in each spacecraft without submitting to the upper layer, thus increasing efficiency. While there are several onboard subnetworks inside a single spacecraft, data routing among different subnetworks can also be implemented in the transfer layer.

f) Services are combined to achieve system functionality.

The various functions of CAST FUHSI can be achieved through a combination of a variety of services. For example, telemetry acquisition in telemetry can apply DDPS in the Application Support Layer to access the device data via DAS or DVS, SPP, Subnetwork Layer Packet Services, or Subnetwork Layer Memory Access Services.

Telemetry organization can generate telemetry packets using the PUS Housekeeping & Diagnostic Parameter Reporting Service in the Application Support Layer. AOS protocol can be used to complete the frame organization and virtual channel scheduling and download the telemetry data to the ground via hardware device/link. These basic services will greatly facilitate the development of various functions of the system, and other subsequent intelligent functions can be developed based on the above basic services, thus reducing the development workload of the user.

7.3 THE CHANGE OF SOFTWARE DEVELOPMENT MODEL

CAST FUHSI offers the standard services in a layered structure, building a comprehensive avionics system software framework and basic service platform. Under the support of standard services, protocols, and interfaces, the software can be used in most spacecraft, and can be used for future intelligent and internetworking applications. A large number of repetitions in the design, implementation, and testing can be avoided by applying the architecture, which can effectively reduce the cost and risk of a space mission.

Because a large number of CCSDS standards have been adopted and implemented as software components, the whole software development mode of the flight software will be fundamentally changed to the assembly model based on software architecture and components. The efficiency of software development and the reliability of software will be improved remarkably in the following several aspects:

a) Software requirement analysis phase

The main work is to select the desired services and protocols from the service and protocol architecture according to the special needs of different spacecraft in the selection of bus, protocol, service, and hardware configuration. Then the parameters are configured according to the requirement, and the mission-specific services and protocols are proposed for the spacecraft when necessary.

Thanks to the use of the CCSDS standard services and protocols, definition activities can be reduced, making the software requirement analysis of the spacecraft more focused on the mission-specific requirements.

b) Software design phase

In this phase, the main work task is the selection of services and protocols as well as their software components in the software architecture, based on the requirement analysis. The components will be tailored, and the special components associated with the application of the task should be designed with the interfaces by which they can be connected to the common software components.

For the spacecraft flight software based on this software architecture, in the process of software design and development as well as software use, the main work changes from software programming to design and configuration of parameters in the standards and services. Each CCSDS service contains a large number of descriptive parameters for the properties and running rules. According to the functional

requirements, the flight software, hardware environment, as well as the attributes and requirements of devices and its users, the user may make the installation and configuration of software components and the initialization parameters by the rules of unified naming rules in the global setting. If using the assembly and simulation software component verification tool, the system information flow and performance can be simulated; this may help to modify the component's configuration and connection.

Supported by the standardized software architecture, users can focus on standard software component configuration and assembly, with no need for repetitive software design. Through layered structure and repeated use of the standard services and protocols, complexity of the verification can be reduced, and the reliability can be improved continuously.

c) Software implementation phase

Because most requirements can be satisfied by a combination of standard CCSDS services and protocols that have already been implemented by software components, only a small amount of software components related to special requirements need to be developed while existing software components can be reused.

d) Software testing phase

Unit test of some inherited components can be skipped, and some service test cases can be reused. The tester only needs to design the new cases for the software components' corresponding special requirements of spacecraft. Hence, the test workload can be greatly reduced.

e) Software maintenance phase

Because of the standard services and protocols in the hierarchical structure, the changing, replacing, or modification of some services or functions will not affect other layers, which is convenient for the upgrade and maintenance of flight software.

7.4 CONCLUSION

CAST FUHSI based on the CCSDS standard greatly enhances its function compared with the traditional spacecraft software system and obviously improves standardization, flexibility, expansibility, and reliability.

a) **Standardization**: The hardware and software functions of each layer are defined by a set of standard services. The definition of services uses CCSDS standards, thereby reducing the demand for the definition activities and achieving the reuse of devices and software as well as transplantation and interoperability. It can meet future application requirements as well as facilitate exchanges and cooperation.

b) **Flexibility**: As a layered protocol and software architecture, the system offers more flexibility for the information transfer mechanism to support the device transmit data through any interface accessing the system, and also supports users in sending information on demand.

c) **Scalability**: Through the component interface design, the interface can be replaced and expanded. For example, when the device access interface of 1553B bus is replaced by a serial port, the original 1553B bus convergence component can be replaced with a serial convergence component, and the software interface of the Subnetwork Layer Packet Service can be kept unchanged.

d) **Reliability**: On one hand, through the joint design of hardware and software, hardware can support interface redirection, and software can achieve task migration and system reconfiguration to enhance the overall reliability of the system. On the other hand, through layering, as well as testing and reusing of the standard services and protocols, the complexity of system verification can be reduced, and the reliability can be continually improved.

ANNEX A

PROCESS AND METHOD EXAMPLES FOR IMPLEMENTING CCSDS STANDARD PRIMITIVES

(INFORMATIVE)

In the CCSDS standard, the external interfaces are mainly described by primitives. The specific implementation processes of the primitives are not described in the CCSDS standard. This architecture uses a number of CCSDS standards. The implementation of the primitives and the way of providing the outside interface is given in detail.

A dynamic data flow analysis method is used in the actual design process. It can not only represent the static processing relation in the conventional data flow diagram, but can also represent the dynamic execution and interaction process, which can be verified by the design of test cases in the requirement phase.

The rules of the data flow diagram are as follows:

a) Its data processing, data storage, data flow, and data representation are consistent with the conventional data flow diagram.

b) Several thick line arrows with step numbers 1, 2, 3, etc., are added in the diagram to represent the execution steps.

c) Address identifiers for the step execution are added in the diagram. S represents that it is executed in the source end; D represents that it is executed in the destination end.

d) In the description of the data flow diagram, its foreground processes are described according to steps for each of the primitives, and its background processes are described if necessary.

e) It is described according to the order of input, processing and output for each process in the data flow diagram.

Taking the SPP in Transfer Layer as an example, the specific implementation process of the method is given.

According to CCSDS SPP, the external interface requirements of the protocol include:

a) PACKET.request: the upper layer requests to send a space packet to the destination via the Transfer Layer.

b) PACKET.indication: the Transfer Layer delivers a space packet to the upper layer.

In view of the above requirements, the implementation process of the PACKET.request primitive can be drawn as shown in figure A-1.

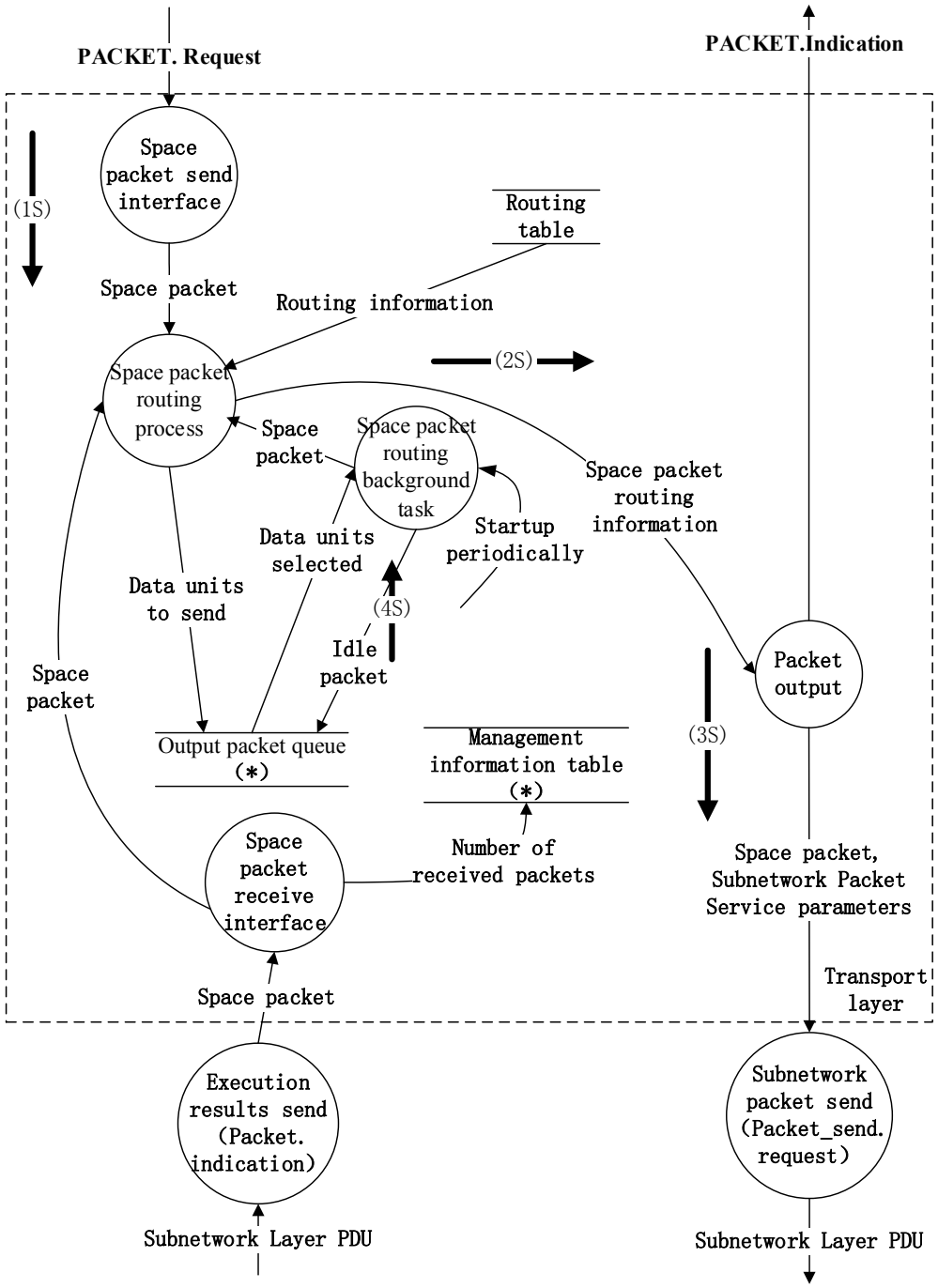

Figure A-1: Implementation Process of PACKET.request Primitive

(1S), (2S), and (3S) are the steps in which the user sends a space packet to the bottom layer with the PACKET.request primitive. The steps are described as follows (for space considerations, some algorithms, parameters, and details of the process are omitted here):

The foreground execution processes of the source end are as follows:

(1S) The space packet sending interface accepts the upper user's call, and then sends the packet to the space packet routing process.

(2S) The space packet routing process queries the routing table to get the routing information. The space packet and routing information are sent to the packet output process.

(3S) Packet output calls the underlying Subnetwork Layer Packet Service primitives to send data.

The background execution processes of the source end are as follows:

(4S) The space packet routing background task periodically takes out data units from the queue, and sends them to packet output.

PACKET.indication primitive is the execution process of the destination end. It can also be described in a similar way, which is not repeated here. According to the above analysis process, one can get the specific implementation process of the PACKET.request primitive, and one can also get the implementation process of the PACKET.indication primitive in a similar way.

ANNEX B

ABBREVIATIONS AND ACRONYMS

(INFORMATIVE)

AMS	Asynchronous Message Service
API	application programming interface
APID	Application Process Identifier
AN	analogue
BSP	board support package
CAST	China Academy of Space Technology
CCSDS	Consultative Committee for Space Data Systems
CFDP	CCSDS File Delivery Protocol
CPU	central processor unit
DACP	Device Abstraction Control Procedure
DAP	Device-specific Access Protocol
DAS	Device Access Service
DDPS	Device Data Pooling Service
DES	Device Enumeration Service
DVS	Device Virtualization Service
DS	digital serial
ECSS	European Cooperation for Space Standardization
EDS	Electronic Data Sheet
FUHSI	Flexible and Unified Flight Software Architecture
IETF	Internet Engineering Task Force
MAPP	Multiplexer Access Point Packet
MAS	Memory Access Service
MASAP	Memory Access Service Access Point
MIB	management information base
ML	memory load
MTS	Message Transfer Service
OBDH	onboard data handling

OSI	Open Systems Interconnection
PDU	protocol data unit
PUS	Packet Utilization Standard
PS	Packet Service
PSSAP	Packet Source Service Access Point
PDSAP	Packet Destination Service Access Point
QoS	quality of service
RAM	random-access memory
ROM	read-only memory
RT	remote terminal
SAP	service access point
SCPS-TP	Space Communications Protocol Standards Transport Protocol
SEDS	SOIS EDS
SIS	Space Internetworking Service
SLS	Space Link Service
SDIU	Spacecraft Data Interface Unit
SDU	service data unit
SMU	Spacecraft Management Unit
SOIS	Spacecraft Onboard Interface Services
SYNC	Synchronisation Service
TAS	Time Access Service
TC	telecommand
TCP	Transmission Control Protocol
TM	telemetry
TTE	time triggered Ethernet
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
USLP	Unified Space Link Protocol

ANNEX C

DESCRIPTION OF THE PARAMETERS BY SEDS

(INFORMATIVE)

```
<?xml version="1.0" encoding="UTF-8"?>
<DataSheet xmlns="http://www.ccsds.org/schema/sois/seds" xmlns:xi="http://www.w3.org/2001/XInclude"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ccsds.org/schema/sois/seds seds.xsd">
  <!-- Include the CCSDS SOIS Subnetwork Service definitions -->
  <!-- Note that the XPointer scheme used here (element) is quite restrictive, but it parses properly with the tools
I am using (XMLSpy). It should also parse OK with most other tools (I know it works with JAXB). -->
  <xi:include href="ccsds.sois.subnetwork.xml" xpointer="element(/1/1)"/>
  <!-- All the types that are necessary for this device are in a specific namespace to help separate things -->
  <Namespace name="DemoML">
    <!-- This is the set of all parameter types which are used in the public interfaces to the component types
described in this namespace -->
    <DataTypeSet>
      <!-- data types from here on -->
      <BooleanDataType name="bool"/>
      <IntegerDataType name="uint8_t">
        <Range>
          <MinMaxRange min="0" max="255" rangeType="inclusiveMinInclusiveMax"/>
        </Range>
      </IntegerDataType>
      <IntegerDataType name="uint16_t">
        <Range>
          <MinMaxRange min="0" max="65535" rangeType="inclusiveMinInclusiveMax"/>
        </Range>
      </IntegerDataType>
      <IntegerDataType name="uint32_t">
        <Range>
          <MinMaxRange min="0" max="4294967295" rangeType="inclusiveMinInclusiveMax"/>
        </Range>
      </IntegerDataType>
      <IntegerDataType name="uint8_t*">
        <LongDescription>it is  a address whose length is 32 bit</LongDescription>
        <Range>
          <MinMaxRange min="0" max="4294967295" rangeType="inclusiveMinInclusiveMax"/>
        </Range>
      </IntegerDataType>
      <IntegerDataType name="dcl_ml_com_t *">
        <LongDescription>it is  a address point</LongDescription>
        <Range>
          <MinMaxRange min="0" max="4294967295" rangeType="inclusiveMinInclusiveMax"/>
        </Range>
      </IntegerDataType>
      <ContainerDataType name="device_access_type_table_1">
        <EntryList>
          <FixedValueEntry fixedValue="8" name="device_id" type="uint16_t"/>
          <FixedValueEntry fixedValue="24" name="cor_dap" type="uint16_t"/>
        </EntryList>
```

```
    </ContainerDataType>
   <ContainerDataType name="device_access_type_table_2">
     <EntryList>
      <FixedValueEntry fixedValue="9" name="device_id" type="uint16_t"/>
      <FixedValueEntry fixedValue="24" name="cor_dap" type="uint16_t"/>
     </EntryList>
    </ContainerDataType>

   <ContainerDataType name="device_value_table">
     <EntryList>
      <FixedValueEntry fixedValue="8" name="device_id" type="uint16_t"/>
      <FixedValueEntry fixedValue="0" name="value_id" type="uint16_t"/>
      <FixedValueEntry fixedValue="7" name="net_addr" type="uint16_t"/>
                    <FixedValueEntry fixedValue="0" name="start_addr" type="uint32_t"/>
      <FixedValueEntry fixedValue="1000" name="dv_length" type="uint16_t"/>
     </EntryList>
    </ContainerDataType>

   <ContainerDataType name="routing_table_1">
     <EntryList>
      <FixedValueEntry fixedValue="0x420" name="net_addr" type="uint16_t"/>
      <FixedValueEntry fixedValue="0x7e0" name="ro_mask" type="uint16_t"/>
      <FixedValueEntry fixedValue="0" name="next_subnet_id" type="uint16_t"/>
                    <FixedValueEntry fixedValue="0" name="next_subnet_addr" type="uint16_t"/>
      <FixedValueEntry fixedValue="0"  name="ass_parameter" type="uint32_t"/>
     </EntryList>
    </ContainerDataType>
   <ContainerDataType name="routing_table_2">
     <EntryList>
      <FixedValueEntry fixedValue="0x8" name="net_addr" type="uint16_t"/>
      <FixedValueEntry fixedValue="0x7ff" name="ro_mask" type="uint16_t"/>
      <FixedValueEntry fixedValue="6" name="next_subnet_id" type="uint16_t"/>
                    <FixedValueEntry fixedValue="0" name="next_subne_addr" type="uint16_t"/>
      <FixedValueEntry fixedValue="0" name="ass_parameter" type="uint32_t"/>
     </EntryList>
    </ContainerDataType>

   <ContainerDataType name="ml_link_table_1">
     <EntryList>
      <FixedValueEntry fixedValue="6" Entry name="link_id" type="uint16_t"/>
      <FixedValueEntry fixedValue="0" Entry name="link_type" type="uint16_t"/>
                    <FixedValueEntry fixedValue="3" Entry name="driver_master" type="uint32_t"/>
      <FixedValueEntry fixedValue="1" Entry name="driver_slave" type="uint32_t"/>
     </EntryList>
    </ContainerDataType>

    <ContainerDataType name="ml_link_table_2">
     <EntryList>
      <FixedValueEntry fixedValue="7" Entry name="link_id" type="uint16_t"/>
      <FixedValueEntry fixedValue="0" Entry name="link_type" type="uint16_t"/>
                    <FixedValueEntry fixedValue="3" Entry name="driver_master" type="uint32_t"/>
      <FixedValueEntry fixedValue="2" Entry name="driver_slave" type="uint32_t"/>
     </EntryList>
    </ContainerDataType>

   </Namespace>
</DataSheet>
```

ANNEX D

DESCRIPTION OF THE INTERFACES BY SEDS

(INFORMATIVE)

```
 <!-- This is the set of all interface types used by component types in this namespace -->
  <DeclaredInterfaceSet>
   <Interface name="tpPacketSend_funcp">
    <ParameterSet>
                        <Parameter name="src_apid" readOnly="true" type="uint16_t" mode="async" />
                        <Parameter name="dest_apid" readOnly="true" type="uint16_t" mode="async" />
                        <Parameter      name="packet_buffer_p"      readOnly="true"      type="uint8_t*"
mode="async" />
                        <Parameter name="length" readOnly="true" type="uint32_t" mode="async" />
                        <Parameter name="qos" readOnly="true" type="uint32_t" mode="async" />
    </ParameterSet>
   </Interface>
  </DeclaredInterfaceSet>
  <DeclaredInterfaceSet>
   <Interface name="tpPacketSend">
    <ParameterSet>
                        <Parameter name="src_apid" readOnly="true" type="uint16_t" mode="async" />
                        <Parameter name="dest_apid" readOnly="true" type="uint16_t" mode="async" />
                        <Parameter      name="packet_buffer_p"      readOnly="true"      type="uint8_t*"
mode="async" />
                        <Parameter name="length" readOnly="true" type="uint32_t" mode="async" />
                        <Parameter name="qos" readOnly="true" type="uint32_t" mode="async" />
    </ParameterSet>
   </Interface>
  </DeclaredInterfaceSet>
  <DeclaredInterfaceSet>
   <Interface name="snPsSend_funcp">
    <ParameterSet>
                        <Parameter name="qos" readOnly="true" type="uint8_t" mode="async" />
                        <Parameter name="priority" readOnly="true" type="uint8_t" mode="async" />
                        <Parameter name="channel" readOnly="true" type="uint8_t" mode="async" />
                        <Parameter name="next_link_id" readOnly="true" type="uint8_t" mode="async" />
                        <Parameter      name="next_sn_address"      readOnly="true"      type="uint8_t"
mode="async" />
                        <Parameter      name="packet_buffer_p"      readOnly="true"      type="uint8_t*"
mode="async" />
                        <Parameter name="length" readOnly="true" type="uint32_t" mode="async" />
    </ParameterSet>
   </Interface>
  </DeclaredInterfaceSet>
  <DeclaredInterfaceSet>
   <Interface name="snPsSend">
    <ParameterSet>
                        <Parameter name="qos" readOnly="true" type="uint8_t" mode="async" />
                        <Parameter name="priority" readOnly="true" type="uint8_t" mode="async" />
                        <Parameter name="channel" readOnly="true" type="uint8_t" mode="async" />
```

```
                        <Parameter name="next_link_id" readOnly="true" type="uint8_t" mode="async" />
                        <Parameter name="next_sn_address" readOnly="true" type="uint8_t"
mode="async" />
                        <Parameter name="packet_buffer_p" readOnly="true" type="uint8_t*"
mode="async" />
                        <Parameter name="length" readOnly="true" type="uint32_t" mode="async" />
      </ParameterSet>
    </Interface>
  </DeclaredInterfaceSet>
  <DeclaredInterfaceSet>
    <Interface name="snDclMLInterface_funcp">
      <ParameterSet>
                        <Parameter name="obj_p" readOnly="true" type="dcl_ml_com_t*" mode="async"
/>
                        <Parameter name="priority" readOnly="true" type="uint8_t" mode="async" />
                        <Parameter name="length" readOnly="true" type="uint32_t" mode="async" />
                        <Parameter name="packet_buffer_p" readOnly="true" type="uint8_t*"
mode="async" />
      </ParameterSet>
    </Interface>
  </DeclaredInterfaceSet>
  <DeclaredInterfaceSet>
    <Interface name=" snDclMLInterface">
      <ParameterSet>
                        <Parameter name="obj_p" readOnly="true" type="dcl_ml_com_t*" mode="async"
/>
                        <Parameter name="priority" readOnly="true" type="uint8_t" mode="async" />
                        <Parameter name="length" readOnly="true" type="uint32_t" mode="async" />
                        <Parameter name="packet_buffer_p" readOnly="true" type="uint8_t*"
mode="async" />
      </ParameterSet>
    </Interface>
  </DeclaredInterfaceSet>
```