

Research and Development for  
Space Data System Standards

|  |
| --- |
| CAST FLIGHT SOFTWARE AS A CCSDS ONBOARD REFERENCE ARCHITECTURE |

EXPERIMENTAL SPECIFICATION

CCSDS TBD.0-O-0

ORANGE BOOK

February 2021

AUTHORITY

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | Issue: | Orange Book, Issue 1 |  |
|  | Date: | February 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

E-mail: 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 nondiscriminatory. 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 e-mail 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 Federal Science Policy Office (BFSPO)/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.
* 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.
* Indian Space Research Organization (ISRO)/India.
* Institute of Space Research (IKI)/Russian Federation.
* KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
* Korea Aerospace Research Institute (KARI)/Korea.
* Ministry of Communications (MOC)/Israel.
* 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.
* 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 TBD.0-O-0 | CAST Flight Software as a CCSDS Onboard Reference Architecture, Experimental Specification, Issue 1 | February 2021 | Original issue |

CONTENTS

Section Page

1 INTRODUCTION 1-1

1.1 PURPOSE 1-1

1.2 SCOPE 1-1

1.3 DEFINITIONS AND CONVENTIONS 1-3

1.4 DOCOMENT STRUCTURE 1-5

1.5 REFERENCES 1-5

2 OVERVIEW OF CAST FLIGHT SOFTWARE ARCHITECTURE 2-1

2.1 BACKGROUND 2-1

2.2 SOFTWARE ARCHITECTURE 2-2

2.3 INTERFACES 2-5

3 INFUSION OF SERVICES AND PROTOCOLS STANDARDS INTO CAST SOFTWARE ARCHITECTURE 3-1

3.1 GENERAL 3-1

3.2 SERVICE AND PROTOCOL ARCHITECTURE 3-1

3.3 RELATIONSHIP BETWEEN SOIS AND OTHER STANDARDS 3-9

4 RELATIONSHIP BETWEEN SOIS SERVICES 4-1

4.1 GENERAL 4-1

4.2 NAMING MECHANISM 4-1

4.3 MAJOR SERVICES RELATIONSHIP AND ADDRESSING MECHANISM 4-3

5 RELATIONSHIP BETWEEN SOIS SERVICES AND DEVICE HARDWARE 5-1

5.1 GENERAL 5-1

5.2 DEVICE TYPES ANALYSIS IN CAST AVIONICS SYSTEM 5-1

5.3 HARDWARE NODES ACCESS METHODS IN AVIONICS SYSTEM 5-3

6 APPLICATION OF SEDS 6-1

6.1 GENERAL 6-1

6.2 AN APPLICATION EXAMPLE 6-1

7 BENEFITS OF USING STANDARDIZED PROTOCOLS AND SERVICES IN CAST SOFTWARE 7-1

7.1 BRIEF INTRODUCTION OF IMPLEMENTION AND EXPERIMENTATION 7-1

7.2 SYSTEM FUNCTION ENHANCEMENT 7-2

7.3 THE CHANGE OF SOFTWARE DEVELOPMENT MODEL 7-4

7.4 CONLUSION 7-6

ANNEX A PROCESS AND METHOD EXAMPLES FOR IMPLIMENTING CCSDS STANDARD PRIMITIVES [INFORMATIVE] A-1

ANNEX B ABBREVIATIONS AND ACRONYMS [INFORMATIVE] B-1

ANNEX C DESCRIPTION OF THE PARAMETERS BY SEDS [INFORMATIVE] C-1

ANNEX D DESCRIPTION OF THE INTERFACES BY SEDS [INFORMATIVE] D-1

CONTENTS (continued)

Figure Page

[1-1: Bit Numbering Convention 1-5](#_Toc63283394)

[2-1: CAST Flight Software Architecture 2-3](#_Toc63283395)

[3-1: CAST Flight Software Service and Protocol Architecture 3-7](#_Toc63283396)

[3-2: Relationship between Service and Protocol Architecture and Software Architecture 3-8](#_Toc63283397)

[3-3: Example of Protocol Configuration 3-16](#_Toc63283398)

[4-1: The Hierarchy of Naming 4-2](#_Toc63283399)

[5-1: Hardware Platform Composition Diagram of Avionics System 5-2](#_Toc63283400)

[5-2: Protocol Configuration of Intelligent Nodes through 1553B 5-3](#_Toc63283401)

[5-3: Protocol Configuration of Intelligent Nodes through TTE 5-5](#_Toc63283402)

[5-4: Protocol Configuration of Accessing Simple Intelligent Nodes 5-6](#_Toc63283403)

[5-5: Protocol Configuration of Accessing Non-Intelligent Nodes 5-7](#_Toc63283404)

[6-1: Sending a ML Command 6-2](#_Toc63283405)

[7-1: Avionics System Hardware Platform and Test System 1 7-1](#_Toc63283406)

[7-2: Avionics System Hardware Platform and Test System 2 7-2](#_Toc63283407)

[A-1: Implementation Process of PACKET.request Primitive A-2](#_Toc63283408)

Table Page

[3-1: Common Functions Mapping to Services and Protocols 3-5](#_Toc55320316)

[3-2: Settings of Para\_id 3-10](#_Toc55320317)

[3-3: Settings of Register Address 3-13](#_Toc55320318)

[3-4: Example of Register Load Command Channel Identifier 3-13](#_Toc55320319)

[3-5: Virtual Device Table 3-13](#_Toc55320320)

[3-6: Virtual Value Resolution Table of the Register Load Command Virtual Devices 3-13](#_Toc55320321)

[3-7: Lookup Table for Virtual Device ID and Virtual Value ID 3-14](#_Toc55320322)

[6-1: Device Access Type Table 6-2](#_Toc55320323)

[6-2: Device and Value Identifier Resolution Table 6-3](#_Toc55320324)

[6-3: Routing Table 6-3](#_Toc55320325)

[6-4: ML Link Configuration Information 6-3](#_Toc55320326)

# INTRODUCTION

## 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 system. Meanwhile, 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.

## SCOPE

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

1. How to integrate the standards of CCSDS SOIS, SLS, SIS and ECSS, IETF in the flight software architecture; and how to set up the interface between SOIS services and other standards;
2. How to establish the functional connection among SOIS services in the flight software architecture;
3. How to connect the standard SOIS services to the specific devices in the flight software architecture;
4. How to apply SEDS in the flight software architecture；
5. The benefits of applying standards in the flight software architecture.

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

1. SOIS Subnetwork Packet Service (Reference [2]);
2. SOIS Subnetwork Memory Access Service (Reference [3]);
3. SOIS Subnetwork Synchronisation Service (Reference [4]);
4. SOIS Message Transfer Service (Reference [5]);
5. SOIS Device Access Service (Reference [6]);
6. SOIS Device Virtualization Service (Reference [7]);
7. SOIS Device Data Pooling Service (Reference [8]);
8. SOIS Time Access Service (Reference [9]);
9. TC Space Data Link Protocol (Reference [10]);
10. AOS Space Data Link Protocol (Reference [11]);
11. Space Packet Protocol (Reference [12]);
12. Communications Operation Procedure-1 (Reference [31]);
13. Encapsulation Service (Reference [36]);
14. Asynchronous Message Services (Reference [13]);
15. 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：

1. SOIS File & Packet Store Service;
2. SOIS Device Enumeration Service;
3. SOIS Device Discovery Service;
4. SOIS Test Service.

Some ECSS standards are also used in the architecture：

1. Packet Utilization Standard (Reference [14]);
2. 1553B Bus Standard (Reference [15]).

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

1. Internet Protocol (Reference [18]);
2. Transmission Control Protocol (Reference [19]).
3. User Datagram Protocol (Reference [20]).

## DEFINITIONS AND CONVENTIONS

### DEFINITIONS

#### Definitions from the Open Systems Interconnection (OSI) Basic Reference Model

1. 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, i.e., 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:
2. entity;
3. service;
4. service-access-point, SAP;
5. service-data-unit, SDU;
6. protocol-data-unit, PDU;
7. service user;
8. service provider;
9. application entity;
10. application layer.

#### 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, i.e., 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:

1. best effort；
2. data link；
3. data system；
4. data system address；
5. device；
6. device abstraction control procedure, DACP；
7. device-specific access protocol, DAP；
8. electronic data sheet, EDS；
9. functional interface；
10. heterogeneous network；
11. packet；
12. protocol ID；
13. quality of service, QoS；
14. reliability；
15. service class；
16. subnetwork；
17. user；
18. virtual device.

#### Terms Defined in This Experimental Specification

1. Software Component: The functional interface, the program code, data and internal variables, etc. are packaged into an independent atomic software unit, which is configurable with the separation of the external environment, called the software components.
2. Transfer layer: It contains the CCSDS transport layer and the network layer, providing a standard interface of data transmission for the upper services and users.
3. Subnetwork layer: It provides a unified interface to the upper layer and shields the difference of various underlying data links, under the transfer layer.

### 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, i.e., ‘Bit 0’.

In accordance with standard data-communications practice, data fields are often grouped into 8-bit ‘words’ which 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’.

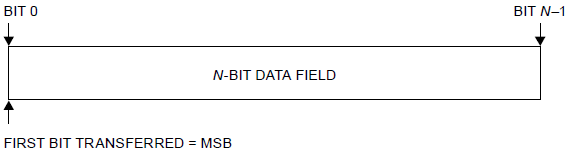


Figure 1-1: Bit Numbering Convention

## DOCOMENT 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.

## 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. Recommendation for 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 Standards (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 Standards (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 Standards (Magenta Book). CCSDS 853.0-M-1. Washington, D.C.: CCSDS, December 2009.
5. Spacecraft Onboard Interface Services--Message Transfer Service. Issue 1. Recommendation for Space Data System Standards (Silver Book). CCSDS 875.0-M-1-S. Washington, D.C.: CCSDS, November 2012.
6. Spacecraft Onboard Interface Services--Device Access Service. Issue 1. Recommendation for Space Data System Standards (Silver Book). CCSDS 871.0-M-1-S. Washington, D.C.: CCSDS, March 2013.
7. Spacecraft Onboard Interface Services--Device Virtualization Service. Issue 1. Recommendation for Space Data System Standards (Silver Book). CCSDS 871.2-M-1-S. Washington, D.C.: CCSDS, March 2014.
8. Spacecraft Onboard Interface Services--Device Data Pooling Service. Issue 1. Recommendation for Space Data System Standards (Silver Book). CCSDS 871.1-M-1-S. Washington, D.C.: CCSDS, November 2012.
9. Spacecraft Onboard Interface Services—Time Access Service. Issue 1. Recommendation for Space Data System Standards (Silver Book). CCSDS 872.0-M-1-S. Washington, D.C.: CCSDS, January 2011.
10. TC space data link protocol. Issue 3. Recommendation for Space Data System Standards (Blue Book). CCSDS 232.0-B-3. Washington, D.C.: CCSDS, September 2015.
11. AOS space data link protocol. Issue 3. Recommendation for Space Data System Standards (Blue Book). CCSDS 732.0-B-3. Washington, D.C.: CCSDS, September 2015.
12. Space packet protocol. Issue 1. Recommendation for Space Data System Standards (Blue Book). CCSDS 133.0-B-1. Washington, D.C.: CCSDS, September 2003.
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. European Cooperation for Space Standardization. ECSS-E-70-41A. Noordwijk: ECSS, January 2003.
15. Space engineering: interface and communication protocol for MIL-STD-1553B data bus on board spacecraft．European Cooperation for Space Standardization. ECSS-E-ST-50-13C. Noordwijk: ECSS, November 2008.
16. TM Synchronization and Channel Coding. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 131.0-B-2. Washington, D.C.: CCSDS, August 2011.
17. TC Synchronization and Channel Coding. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 231.0-B-2. Washington, D.C.: CCSDS September 2010.
18. Internet Protocol Version 6(IPv6) Specification. RFC2460. Reston, Virginia: ISOC, December 1998.
19. Transmission Control Protocol. RFC793. Reston, Virginia: ISOC, September 1981.
20. User Datagram Protocol. RFC768. Reston, Virginia: ISOC, August 1980.
21. TM Space Data Link Protocol. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 132.0-B-2. Washington, D.C.: CCSDS, September 2015.
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 4. Recommendation for Space Data System Standards (Blue Book), CCSDS 727.0-B-4. Washington, D.C.: CCSDS, January 2007.
24. Proximity-1 Space Link Protocol—Data Link Layer. Issue 5. Recommendation for Space Data System Standards (Blue Book), CCSDS 211.0-B-5. Washington, D.C.:CCSDS, December 2013.
25. Proximity-1 Space Link Protocol—Coding and Synchronization Sublayer. Issue 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 211.2-B-2. Washington, D.C.: CCSDS, December 2013.
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 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 121.0-B-2. Washington, D.C.: CCSDS, May 2012.
28. Image Data Compression. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 122.0-B-1. Washington, D.C.: CCSDS, November 2005.
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. <http://sanaregistry.org/r/sois/>.
33. CCSDS SOIS Electronic Data Sheets and Dictionary of Terms for Onboard Devices and Components. Draft Report Concerning Space Data System Standards (Draft Green Book), CCSDS 870.0-G-0. Washington D.C.: CCSDS, June 2017.
34. Spacecraft Onboard Interface Services—Specification for Dictionary of Terms for Electronic Data Sheets for Onboard Components. Recommendation for Space Data System Practices (Blue Book), CCSDS 876.0-B-1. Washington, D.C.: CCSDS, April 2019.
35. IP over CCSDS Space Links. Recommendation for Space Data System Standards (Blue Book). CCSDS 702.1-B-1. Washington, D.C.: CCSDS, September 2012.
36. Encapsulation Service. Recommendation for Space Data System Standards (Blue Book). CCSDS 133.1-B-2. Washington, D.C.: CCSDS, October 2009.

# OVERVIEW OF CAST FLIGHT SOFTWARE ARCHITECTURE

## BACKGROUND

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

1. The intelligence of spacecraft should be enhanced gradually. Functions such as autonomous mission planning，self-determination，autonomous housekeeping need to be implemented by avionics systems. Besides, the computing ability needs to be extended as required.
2. The ability of internetworking needs to be provided, 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 mechanism will be implemented to allow the cooperation of multiple spacecraft or devices inside a spacecraft. Thereby the user can focus on the implementation of algorithm to support intelligentization.
3. 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 affect 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 developing process of application software can be simplified by the integration of common services. Based on the requirements above, multiple domain requirements of spacecraft avionics system have been analyzed by CAST, services and protocols of CCSDS and ECSS have been selected and integrated (as specified in section 3), an avionics system flight software architecture has been designed. The purposes are:

1. providing a standard software platform to support the intelligent applications for future spacecraft, the space internetworking and onboard networking.
2. accelerating the reuse of onboard software, onboard devices and ground test software, in the meantime enhancing the system functions and reducing the repetition of development.
3. 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.

## SOFTWARE ARCHITECTURE

### OVERVIEW

The principles of CAST FUHSI design are as follows:

1. **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 make FUHSI very flexible. On the other hand, the common functions can be provided through standard services, which can increase the reusability of software.
2. **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.
3. **Unified information transfer mechanism:** A unified information transfer mechanism is established based on CCSDS standards, ECSS standards, IETF standards, which support the integrated communications and standardized design over ground-to-spacecraft, onboard spacecraft 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.
4. **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 developing process and shorten the software developing 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 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 on board computer, 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, TTE (Time Trigged Ethernet) interface, extension interface and so on.

The operating system layer is the supporting platform, which 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 is 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 application support layer, the integrated communications over spacecraft-to-ground, onboard spacecraft and spacecraft-to-spacecraft links can be implemented. With the support of 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

### 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. Operating System consists of 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.

### 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:

1. Subnetwork Layer. In this layer, a unified software interface is defined to shield the difference on data links. Besides, 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 TC component，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.
2. 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 UDP component to implement UDP protocol. Network layer includes SPP component, IPv6 component, IP over CCSDS component and encapsulation service component. SPP and encapsulation protocols can be distinguished by the packet version number, and they are compatible in network layer. IP component can work on top of IP over CCSDS component and encapsulation service component.
3. Application Support Layer. This Layer provides the standard service components to support the application, which include the SOIS application support layer services and 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.

### 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 multi-task operating system, it could be several tasks or processes, which use the standard interface provided by middleware layer, to accomplish the specific functions of the mission. The interface of MTS will be used for message communications among tasks or processes.

## INTERFACES

### INTERFACE OF EACH LAYER

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

1. Operating System Layer interface: it includes task management interface, interrupt management interface, memory management interface, semaphore management interface, timer management interface, IO interface, user support library interface and so on.
2. Subnetwork Layerinterface: it includes Packet Service interface, Memory Access Service interface, Synchronization Service interface, TC interface, AOS interface and so on.
3. Transfer Layer interface: it includes SPP interface, UDP interface, IPv6 interface, IP over CCSDS interface, encapsulation service interface and so on.
4. Application Support Layer interface: it includes PUS interface, MTS interface, DDPS interface, DAS interface, DVS interface, TAS interface and so on.

### INTERFACE OF SOFTWARE COMPONENT

The middleware of the software architecture is implemented by software components. The interface of component which adopted by CAST is consist of component inner parameters and outside interface. The outside interface contains the following two types.

1. The provided interface to upper layers, including:
2. Initialization Interface: it can be called by other components to accomplish the initialization process.
3. Functional Interface: it can be called by other components to accomplish the main function of the component.
4. Configuration Interface: it can be called by system configurator to accomplish the configuration of the component.
5. The required interface from lower layers: it can be called by the component, and it can be implemented through configuration.

# INFUSION OF SERVICES AND PROTOCOLS STANDARDS INTO CAST SOFTWARE ARCHITECTURE

## GENERAL

Services and protocols of CCSDS and ECSS have been selected and integrated in CAST FUHSI as specified in section 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, SIS domain as well as PUS and 1553B standard from ECSS. The steps of integration are:

1. analyzing the requirements of avionics systems serving for different types of CAST spacecraft.
2. analyzing the adaptability and applicability of CCSDS standards, ECSS standards and IETF standards.
3. 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:

1. Services and protocols architecture, including requirements analysis, analysis and selection of standard services and protocols, design of services and protocols architecture.
2. Relationship between SOIS and other standards, including relationship between SOIS and PUS, relationship between SOIS and SLS protocol, relationship between SOIS and SIS protocol.

## SERVICE AND PROTOCOL ARCHITECTURE

### 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 concluded the common requirements for avionics systems which 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 7 top-level functions: telecommand management, telemetry management, time management, housekeeping management, thermal control management, power management, unlock and rotation gear control, etc.

1. 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 channel to other application process, etc.
2. 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.
3. 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.
4. 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.
5. Thermal control management includes open loop control, close loop control, failure detection and handling, thermal parameters set, etc.
6. Power management includes electricity adjust, power distribution, coulometer control, battery temperature excess protection, etc.
7. Unlock and rotation gear control includes explosive device control, antenna and solar array driving control, etc.

Besides the common requirements mentioned above, different spacecrafts 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 system, in order to communicate with ground system and other devices. In the meantime, some common services are needed by different functions. For example, command sending service is needed by functions such as telecommand management, housekeeping management, thermal control management, power management, etc. Telemetry data acquiring service is needed by functions such as telemetry management, housekeeping management, thermal control management, power management, etc. Time access service is needed by telemetry and telecommand function. Functions related to intelligence such as autonomous mission planning and self-determination also need to acquire telemetry data and send commands. Besides, different functions need a message transfer service to achieve cooperation.

### SELECTION AND ANALYSIS OF STANDARD SERVICE AND PROTOCOL

Fundamental functions such as telemetry data acquiring, command sending, message transfer and time access mentioned in section 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. Besides, 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.

1. CCSDS space communications protocols are developed by workgroups of SLS and SIS domain. There are 5 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.
2. CCSDS has a standard for physical layer called the Radio Frequency and Modulation Systems, which is mainly related to hardware implementation. So it is not considered in CAST FUHSI.
3. 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，Unified Space Link Protocol（USLP）. As services defined by AOS has 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 uplink protocol. Proximity-1 (Reference [24] [25] [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]).
4. 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 Service (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 Service, IP protocol and IP over CCSDS protocol are used in network layer. As these protocols are used, it is easy to support the extension of ground network to space network.
5. 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.
6. CCSDS onboard communications protocols are developed by workgroups of SOIS. There are three layers in the SOIS reference architecture: Subnetwork layer, Transfer Layer and Application Support Layer. The selection of each layer is as follows:
7. 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.
8. 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 Service, 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.
9. Application Support Layer in SOIS reference model contains Command and Data Acquisition Service (CDAS), TAS, MTS, File and Packet Store Service (FPSS), 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.
10. 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.

### 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 spacecrafts such as remote sensing, navigation, telecommunication, crewed spaceship 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** |
|  | Telemetry management | MTS, DDPS, DAS, DVS, TAS, PS, MAS, SYNC | TC, COP-1（for earth orbit only）, AOS, SPP, Encapsulation Service | 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 |
|  | Telecommand management | MTS, DAS, DVS, TAS, PS, MAS, SYNC | TC, COP-1, AOS, SPP, Encapsulation Service | AMS, UDP, IPv6, IP over CCSDS | PUS Telecommand Verification Service, PUS Device command distribution service, PUS Onboard operations scheduling service, ECSS 1553B |
|  | Housekeeping management | MTS, DDPS, DAS, DVS, TAS, PS, MAS, SYNC | TC, COP-1, AOS, SPP, Encapsulation Service | AMS, UDP, IPv6, IP over CCSDS | PUS Event reporting service, PUS Onboard monitoring service, PUS event-action service, PUS Memory management service, ECSS 1553B |
|  | Time management | MTS, DAS, DVS, TAS, PS, MAS, SYNC | TC, COP-1, AOS, SPP, Encapsulation Service | AMS, UDP, IPv6, IP over CCSDS | PUS Time management service, ECSS 1553B |
|  | Thermal control management | MTS, DDPS, DAS, DVS, PS, MAS | TC, COP-1, AOS, SPP, Encapsulation Service | 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 |
|  | Power management | MTS, DDPS, DAS, DVS, PS, MAS | TC, COP-1, AOS, SPP, Encapsulation Service | 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 |
|  | Unlock and rotation gear control | MTS, DDPS, DAS, DVS, PS, MAS | TC, COP-1, AOS, SPP, Encapsulation Service | 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 the 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:

1. Application Layer

Application Layer consists of Application Management Layer and Application Support Layer. 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.

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

1. Transfer Layer

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

1. 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 to extend to support other buses and interfaces.

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

## RELATIONSHIP BETWEEN SOIS AND OTHER STANDARDS

### GENERAL

As specified in section 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, which includes:

1. relationship between SOIS services and PUS services;
2. relationship between SOIS services and SLS protocols;
3. relationship between SOIS services and SIS protocols.

### RELATIONSHIP BETWEEN SOIS SERVICES AND PUS SERVICES

#### 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:

1. DDPS is used to acquire data;
2. DVS is used to send command;
3. MTS is used to send and receive packets;
4. TAS is used to get onboard time.

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

#### PUS Onboard Monitoring Service

PUS onboard monitoring service is used to automatically monitor types of onboard specific parameters, and generates 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:

1. 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, the parameter Acquisition Interval in the primitive corresponds to the parameter monitoring interval;
2. DDPS issues ADD\_ACQUISITION\_ORDER.indication primitive to return the Acquisition Order Identifier to the Onboard monitoring service;
3. Onboard monitoring service calls START\_ACQUISITIONS.request primitive, using the Acquisition Order Identifier to start the acquisition;
4. 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;
5. After DDPS accomplishes the acquisition, if the Asynchronous Acquisition Indication Flag is set in the order, then issues an ACQUISITION.indication primitive to on board monitoring service;
6. When the indication is received or the running cycle is due, on board monitoring service will use a READ\_SAMPLES.request primitive to acquire the data samples;
7. Onboard monitoring service issues a READ\_SAMPLES.indication primitive to deliver the Samples and Result Metadata to on board monitoring service;
8. 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 the 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 the 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 bit 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, 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.

#### 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.

1. Onboard operations scheduling service (application process APID\_A) calls Register.request primitive of MTS to complete the registration;
2. Telecommand packet destination application process APID\_B uses the same way to complete registration, and sends a Assert\_invitation.request invitation with the subject of command message;
3. Application process APID\_A replies Assert\_invitation.indication to application process APID\_B, to agree the invitation;
4. Onboard operations scheduling service runs periodically, calling the TIME.request primitive of TAS to get onboard time; TAS issues TIME.indication to return the onboard time to onboard operations scheduling service;
5. Onboard operations scheduling service compares the onboard time with the Abs/Rel Time Tag in the enabled sub-schedule, 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.
6. The destination application process APID\_B receives the telecommand packet through Message.indication primitive of MTS, and performs the further processing.

In the 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.

#### 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:

1. 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 is 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.
2. 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).
3. 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 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 2 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 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 loadcommand virtual devices) | 2 | Virtual value resolutiontable address of the register load command virtual devices |

Virtual value resolutiontable 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: Lookup Table for Virtual Device ID and Virtual Value ID

|  |  |  |
| --- | --- | --- |
| **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 then are transmitted to DVS together with data. DVS can search the Virtual Device Table according to virtual device ID to get corresponding virtual value resolution table address, and then looks for 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.

### RELATIONSHIP BETWEEN SOIS SERVICES AND SLS PROTOCOLS

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

For telecommand, the application process is as follows:

1. 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.
2. 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 Service 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 above the Transfer Layer, the other refers to other nodes, to which the Subnetwork Layer Packet Service will be used and the node will be responsible to route the packet.

For telemetry, the application process is as follows:

(1) User requests to transfer a space packet using PACKET.request primitive of SPP, together with parameters such as space packet, APID, etc. Besides, user also transfers an encapsulation packet using Encapsulation Service, together with parameters.

(2) 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 Service gets data transmitted to ground, encapsulates the data as packet and calls the PACKET.request primitive of AOS to transfer the packet to AOS protocol entity.

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

### 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 Service, SOIS protocols and PUS protocols of ECSS. Besides, AMS can also be used between two spacecrafts 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:

1. There is no single central node, i.e. Configuration Server of AMS in the spacecraft, all registers have equal authority, which forms distributed network architecture that has no central node.
2. 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 requester and priorities of information.
3. 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.
4. MTS of different devices has no cyclic exchange of Synchronization service, except for initial stage.

### 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:

(1) The SOIS application support layer services such as MTS can use UDP/IP to send and receive messages.

(2) 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

# RELATIONSHIP BETWEEN SOIS SERVICES

## GENERAL

The service and protocol architecture showed in section 3.2.3 use 5 services from SOIS Application Support Layer, SPP in Transfer Layer, and 3 services from 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 application of SOIS services and protocols.

The chapter shall include:

1. naming mechanism;
2. major service relationship and addressing mechanism, which shall include the relationship between MTS and services below, relationship between CDAS and services below, relationship between TAS and services below, relationship between services of Transfer Layer and services of Subnetwork Layer.

## NAMING MECHANISM

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



Figure 4-1: The Hierarchy of Naming

1. APID is used to distinguish the applications in Application Management Layer.
2. In Application Support Layer, Node ID is used to identify the users of 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 Physical Value Identifier.
3. In 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.
4. In Subnetwork Layer, Packet Service has several names, such as Link ID, Subnetwork Address and so on. In application, Link ID together with Subnetwork Address comprises PDSAP address in 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 Subnetwork Address to RT address and one or more RT sub-addresses. Memory Access Service has names including Link ID, Subnetwork Address, Memory ID, Start Memory Address, and so on. Link ID together with Subnetwork Address comprises Destination Address in Memory Access Service primitive. Because the introduction of Transfer Layer, packets transmitted from Transfer Layer to Subnetwork Layer have source address and destination address in it, PSSAP address of Packet Service primitive and 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 mechanism are shown in section 4.3.

## MAJOR SERVICES RELATIONSHIP AND ADDRESSING MECHANISM

### 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:

1. identifying the devices and parameters in the device access request of users;
2. 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;
3. receiving access results, storing them for the user, or sending the results to DAS on the remote device;
4. submitting the access results obtained to the user.

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

In CAST FUHSI, two Device-specific Access Protocol (DAP) types of DAS are further divided, including:

1. 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:
2. 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.
3. 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).
4. 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 device through remote access DAP with 1553B bus.
5. 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:
6. 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.
7. 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 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:

1. 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.
2. 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 Transfer Layer with PACKET.request primitive.
3. Transfer Layer routes packets according to APID, and sends data through Subnetwork Layer Packet Service. The specific process of Transfer Layer is described in section 4.3.4.

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

### 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 AMS standard, 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 Transfer Layer in a unified way, which support SPP and UDP/IP currently 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:

1. After registration and invitation by the receiver, the user sends Send.request to MTS and the destination is identified by APID.
2. 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 not local, PDU, destination APID, and other parameters are sent to Transfer Layer together.
3. According to APID, Transfer Layer can get subnetwork Packet Service parameters, and send data to Packet Service, which sends the data to the destination with the convergence link between processors.
4. The destination receives the data through Subnetwork Layer, and passes the data to Transfer Layer. Transfer Layer sends the data to MTS, which submits the data to the user.

### IDENTIFICATION OF RELATIONSHIPS BETWEEN TIME ACCESS SERVICE AND SERVICES UNDERLYING

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

1. The user invokes TIME.request primitive of TAS;
2. TAS invokes TIME.request primitive of Synchronization Service upon receipt of the request;
3. The TIME.request primitive of Synchronization Service invokes the corresponding device driver of the clock to obtain current spacecraft time and returns the time to TAS through TIME.indication;
4. TAS receives the time and returns the time to the user through its TIME.indication primitive.

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

### IDENTIFICATION OF RELATIONSHIPS BETWEEN TRANSFER LAYER AND SUBNETWORK LAYER

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

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

1. The upper layer service or protocol invokes PACKET.request primitive of Transfer Layer to send data;
2. Transfer Layer routes packets according to APID in the primitive, obtains corresponding Link ID and Subnetwork Address (corresponding to PDSAP address parameter of Subnetwork Layer Packet Service), service type, channel, priority and so on, and invokes PACKET\_SEND.request primitive of Subnetwork Layer Packet Service;
3. Packet Service of 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;
4. 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:

1. 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;
2. 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;
3. 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 Transfer Layer;
4. Transfer Layer forwards the response packets to DAS according to the destination APID.

# RELATIONSHIP BETWEEN SOIS SERVICES AND DEVICE HARDWARE

## GENERAL

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

1. Packet Service in subnetwork layer and convergence layer functions 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.
2. Memory Access Service offers memory read/write operations. It should establish a relationship to these memory operations of actual hardware within the architecture.
3. Synchronization Service in subnetwork is a time-related service, therefore, it should deal with the clock interface of hardware.
4. DAS, DVS and DDPS 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.

## DEVICE TYPES ANALYSIS IN CAST AVIONICS SYSTEM

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

1. Intelligent nodes: these nodes provide strong processing ability and support complete protocol stack with message processing capabilities, which can handle peer-to-peer communication between each other. 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 module of OBDH onboard computer, attitude control computer and payload management computer.
2. Simple intelligent nodes: these nodes are slightly less intelligence 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.
3. 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 are non-intelligent nodes are devices which are attached to On/Off command, AN, DS, ML interface and so on.

The following parts take a design of spacecraft avionics system for example, and demonstrate how those 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:

1. TM/TC interface.
2. command interface: including ONOFF interface, ML interface, etc.
3. data collection interface: including analog collection interface, DS interface and etc.
4. bus interface: including 1553B bus and etc.

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

Other subsystem devices which are connected to 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.

Section 5.3 gives access methods for specific nodes.

## HARDWARE NODES ACCESS METHODS IN AVIONICS SYSTEM

### ACCESS METHODS OF INTELLIGENT NODES

The protocol configuration of two intelligent nodes which are communicated through 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:

1. Application layer: the application process in the two intelligent nodes can subscribe, publish, send messages via the primitives provided by MTS, different nodes can be distinguished by APID;
2. Application Support Layer: MTS uses AMS for implementation, the underlying protocol uses SPP;
3. 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;
4. 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 establish a connection with the hardware is the Subnetwork Layer. Because different data links have its protocols, in order to provide a unified interface to the upper layer, the Packet Service in Subnetwork Layer should provide uniform packet sending interface to the upper layer applications so as to shield the difference of underlying data links. Therefore, the upper layer applications do not need to concern the difference of 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 is sent to the destination or waypoint through convergence layer. And 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 Subnetwork Layer, the convergence layer is a key point. Due to the fact that 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. Packet Service in Subnetwork Layer selects the convergence layer send interface according to the identification passed from 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 its 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 packet. 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 as well as other information, then sends it through the device driver. When segment data arrives, the convergence layer in receiver will read the data through device driver and assemble each segment correspondingly, then commit to upper layer application once the whole packet is received.

The protocol configuration of two intelligent nodes which 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:

1. Application Management layer: the application process in the two intelligent nodes can subscribe, publish, send messages via the primitives provided by MTS;
2. Transport Layer: UDP provides packet transmission to the upper layer and uses the IPv6 as underlying layer protocol to transmit data;
3. Network Layer: IPv6 provides packet transmission to the upper layer and uses the Packet Service as underlying layer service to transmit data on TTE bus;
4. Subnetwork Layer: Data are sent or received through the Packet Service, TTE convergence layer, TTE bus link protocol and physical hardware.

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

### 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:

1. Application layer: the application process in intelligent node can access data from simple intelligent node through CDAS;
2. Application Support Layer: CDAS uses SPP which is provided by underlying layer. For example, DAS of CDAS can be used to send/receive a space packet to simple intelligent nodes;
3. 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;
4. 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 which is implemented in intelligent node, 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 intelligent node asynchronously, its DAP can be configured as packet-send DAP as mentioned in section 4.3.1. And when a simple intelligent node need to receive data from intelligent node, its DAP should be configured as packet-receive DAP as mentioned in section 4.3.1.

### ACCESS METHODS OF NON-INTELLIGENT NODES

The protocol configuration of a non-intelligent node exchanging data with an intelligent node using DS, ML or 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:

1. Application layer: the application process in intelligent node can access data from non-intelligent node through CDAS;
2. Application Support Layer: CDAS uses SPP which is provided by underlying service, thus DAS can send/receive a space packet for instance;
3. 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;
4. Subnetwork Layer: Data are sent or received through 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:

1. Remote Access: through the bus (e.g. 1553B bus) or space link access;
2. Inter-Module Access: through the I/O backplane bus;
3. 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. Target application receives memory access request, and performs intra-module or inter-module access operation, then encapsulates the result into a packet and transmits to the source vice versa.

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

# APPLICATION OF SEDS

## 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 for input of subsequent software testing.

## AN APPLICATION EXAMPLE

In CAST FUHSI, there are 27 software components. We chose a typical example of sending a memory load (ML) command, which runs through the various layers of the architecture, as shown in Figure 6-1. The application of SEDS is illustrated by this example. Sending a ML command involves the following services:  
(1) Device Access Service of the application support layer;  
(2) Space Packet Protocol of the transfer layer;  
(3) Packet Service of the subnetwork layer;  
(4) ML Convergence Service (Convergence\_ML) of the convergence layer.



Figure 6-1: Sending a ML Command

### INSTRUCTION SENDING PROCESS

The ML command is sent from the application layer to the data link layer. The specific sending process is as follows:  
1) Application Management layer: the device access service of the intelligent node accesses the simple intelligent node;  
2) Application support layer: configure 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 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 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 |

3) 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）. Packet is then sent to 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 |

4) 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 |

### PARAMETERS AND INTERFACES TO BE DESCRIBED BY SEDS

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

1) 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.

2) Interface

a) 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)

b) 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)

c）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)

d）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.

# BENEFITS OF USING STANDARDIZED PROTOCOLS AND SERVICES IN CAST SOFTWARE

## BRIEF INTRODUCTION OF IMPLEMENTION 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 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 bring the change of whole software development mode, which can improve 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 equipments and onboard equipments.

Ground equipments include 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), Network Simulator 1 (corresponding to Satellite) and 2 (corresponding to Satellite)。

Onboard equipments include SMU, TTE Switch and Inter-Satellite Router which 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 CCSDS standard can not only provide richer, more practical and more powerful functions than traditional spacecraft software system, but also bring the change of whole software development mode, which can improve efficiency and reliability of software development.

## SYSTEM FUNCTION ENHANCEMENT

System function enhancement mainly manifests in the following aspects:

1. 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 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 then 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.

1. 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 other 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.

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

In CAST FUHSI, the CCSDS MTS, DAS, DVS, DDPS are applied and cooperated with the SPP underlying, the Subnetwork Layer Packet Service as well as 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.

1. If the processes are in the same processor module, the MTS uses local buffer to complete the exchange of messages;
2. 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;
3. 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.

1. 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 the efficiency. The two disadvantages then can be avoided: (a) it no longer needs to segment large data packets into multiple small packets and calculate the address one by one troublesomely; (b) It avoids the low the transmission efficiency caused by waiting for the confirmation of the previous frame before the user sending the next frame.

1. Space and onboard communications are integrated via transfer layer.

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

1. 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 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.

## 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 the 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.

Due to the use of the CCSDS standard services and protocols, we can reduce the definition activities, 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 via 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 the 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 which 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

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

## CONLUSION

CAST FUHSI based on the CCSDS standard greatly enhances its function compared with the traditional spacecraft software system, and obviously improves the 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 flexible for information transfer mechanism to support the device transmit data through any interface to access the system, and also support users to send 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.

1. PROCESS AND METHOD EXAMPLES FOR IMPLIMENTING CCSDS STANDARD PRIMITIVES  
     
   [INFORMATIVE]

In CCSDS standard, the external interfaces are mainly described by primitives. The specific implementation processes of the primitives are not described in CCSDS standard. This architecture uses a number of CCSDS standards. The way to implementing the primitives and the way to 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 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:

1. Its data processing, data storage, data flow and data representation is consistent with the conventional data flow diagram.
2. Several thick line arrows with step numbers 1, 2, 3, etc. are added in the diagram to represent the execution steps.
3. 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.
4. 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.
5. 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:

1. PACKET.request: the upper layer requests to send a space packet to the destination via the Transfer Layer.
2. 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 space packet routing process.

(2S) 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, we can get the specific implementation process of the PACKET.request primitive, and we can also get the implementation process of the PACKET.indication primitive in a similar way.

1. 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 |
| GVCID | Global Virtual Channel Identifier |
| 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 |
| 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 Trigged Ethernet |
| UART | Universal Asynchronous Receiver/Transmitter |
| UDP | User Datagram Protocol |
| USLP | Unified Space Link Protocol |

1. 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>

1. 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>