Design and Implementation of SEDS in Spacecraft Software

Lijun Yang[1], Bohan Chen[1], and Xiongwen He[1,2(✉)]

[1] Beijing Institute of Spacecraft System Engineering, Beijing 10094, China
[2] Tsinghua University, Beijing 10084, China

Abstract. In order to meet the requirements of rapid integration and test of onboard software, the research status of SEDS in the field of CCSDS standard interface service (SIOs) of Space Data System Advisory Committee is analyzed, and this paper discusses how to apply and implement SEDS standard in Chinese spacecraft software. Based on the software layered architecture, the top-level architecture of SEDS reference is designed during the development of the integrated electronic system. Taking the telemetry acquisition function of spacecraft as a use case, this paper designs and implements the application of SEDS and the extension method of SEDS subsequent application. It is verified with SEDS as the input in each stage of software development and testing. With SEDS as the input, the design verification is carried out in each stage of spacecraft software development. The results show that the application of SEDS can realize the standardization of aerospace software data interface and shorten the software development cycle through the reuse of electronic data forms and the automatic generation of relevant codes in each stage of development.

Keywords: Spacecraft software · SEDS · SOIS

1 Introduction

SEDS (SOIS Electronic Data Sheet) is an Electronic Data Sheet used to describe device information and service interface information in the Spacecraft Onboard Interface Services (SIOS) architecture of the Consultative Committee for Space Data Systems (CCSDS).Data description includes: the interface of two-way data exchange between SOIS layers; Service implementation of mapping between two groups of interfaces; The command and parameters that make up the interface; State machines, variables, and behaviors of components; The types, variables, encodings, and terms used as references above.

Based on the requirements of current spacecraft software integration and test cycle shortening, SEDS can describe the equipment information and business interface information, automatically generate onboard software code, test cases and related documents through tools, so as to reduce the time of onboard software integration, test and maintenance, and ensure the consistency of data in each development stage. The implementation and application of this standard can shorten the development cycle, reduce the risk and cost of spacecraft software. This paper first analyzes the current

Q. Guo et al. (Eds.): WiSATS 2021, LNICST 410, pp. 226–236, 2022.
https://doi.org/10.1007/978-3-030-93398-2_23

situation of SEDS research and application abroad, and then designs and implements SEDS application based on hierarchical software architecture.

2 Research Status

At present, SEDS is first applied in NASA and ESA. NASA designed and implemented SEDS in CFE core flight software architecture; ESA is currently in progress, including the development of support related tools.

NASA core software system CFE has been used in many models, based on the software bus as the core, the software components communicate with each other, not only for spacecraft, but also for other aircraft. In the core flight software architecture of NASA's CFE, SEDS electronic data form is used to automatically generate configuration code of onboard software for software assembly. SEDS can be used to define the interface information of devices and software construction services. SEDS can be used to generate software code. On the contrary, SEDS can be generated by scanning the parameter definition or header file of the code. After that, the remote control command code needed for test can be generated by SEDS, which can also be used for telemetry data analysis. NASA has developed tools that use software component SEDS and task configuration files as header files for code generation. The header file contains the message definition and engineering unit transformation. Currently, the tool is integrated into the CFS (core flight system) creation system and used by several NASA centers.

Scisys company in Europe has been researching SEDS and is developing tools to support SEDS. The tool is developed in Java language, and the configuration code can be generated automatically according to SEDS. Its application process is as follows: in the early stage of software development, SEDS file of equipment information can be generated by tools. SEDS file can generate documents of parameter information, test, etc., and can also generate part of component information of software, such as device driver. Then, in software development, when new devices are added or device information is updated, SEDS can be upgraded directly; SEDS can also be generated from the model or parameter information of the software system. When the software system data and equipment information change, the data can be modified through SEDS, and the modified SEDS file can be automatically updated through the tool. EDS can be directly used as its input in comprehensive test, because SEDS directly contains all its document data.

3 SEDS Design

3.1 Top-Level Design

The software architecture based on SOIS is a layered architecture, each layer contains a variety of county protocol and related services. The business and protocol of SOIS standard are encapsulated as reusable software components, and the software components are divided into application layer and middleware layer, which constitute the whole software together with the operating system layer. In the software system,

software architecture, software component forming architecture model and component product model library are used to support software development and SEDS generation; The data dictionary mainly contains the description of semantics, types and structures according to constraints; SEDS library is based on architecture, component library and data dictionary to describe equipment, parameters, interfaces and other information (Fig. 1).

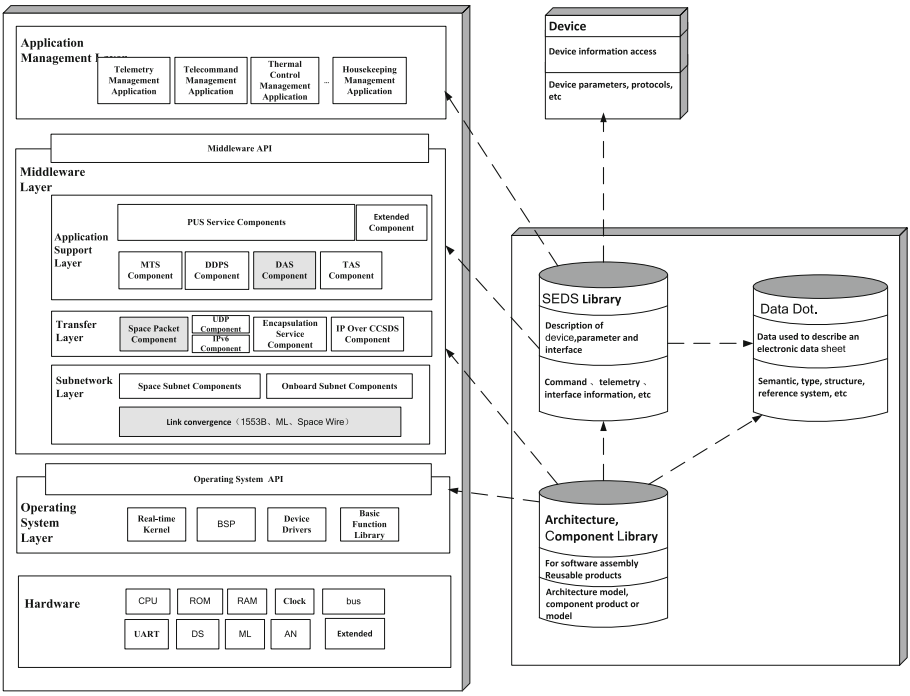

Fig. 1. Top-level design

1) Operating system and hardware layer
   Hardware layer is the bottom layer of software and the basis of software operation. It includes all kinds of hardware of onboard computer, including processor, memory, fixed memory, timer, various interfaces, etc. The interface of the operating system is encapsulated to provide a unified application programming interface for the operating system. SEDS description is not currently involved. SEDS device description mainly refers to the devices connected to CTU and interacting with CTU, not the CTU devices themselves.
2) The middleware layer
   Between the operating system layer and the application layer is the middleware layer, which is mainly a general service system, and its internal is divided into three layers. The levels are as follows:

- Subnetwork layer, a set of service components used to support the upper components, including satellite-borne subnet components, space subnet components, satellite-borne link convergence and so on.
- Transfer layer, transfer layer, mainly used for data transmission, includes the network layer and transport layer of CCSDS. The transport layer includes UDP components and implements the UDP transport protocol.
- Application support layer. The application support layer covers the basic application functions of spacecraft software system, and provides standard services for platform application support.

In the application support layer, transfer layer and subnetwork layer, SEDS is mainly used to describe the interface of components and the description of related configuration parameter tables, such as device table, routing table and so on.

3) The application layer
The application layer is the application software corresponding to the general function, including telecontrol, telemetry, internal management, time management, thermal control management, power management, unlocking and rotation control and so on. Because of the basic service support of the underlying software, the implementation of application layer only needs to combine different basic services according to specific logic. In the application layer, SEDS is mainly used to describe command parameters, telemetry parameters and so on. Once described, these parameters can be used for code generation, developer test, system test and even ground system.

3.2 SEDS Design

Taking the command sending function in spacecraft software as an example, from top to bottom, it corresponds to the software application layer, application support layer, transfer layer, sub-network layer, operating system and hardware layer.D1 and D2 are different devices, D1 is connected to the CTU through the ML channel;D2 is connected to CTU through 422.The ground command is firstly distributed through Telecontrol processing of the application layer, and then data is sent to the space packet protocol of the transfer layer through message sending primitive of the message transmission service of the application support layer. After unpacking, the command are sent to the subnet layer and finally executed by the device.

In this process, SEDS starts from the device. When SEDS describes the device information, since different devices support different data, the SEDS described by the device includes the access interface of the device, the functional interface of the device, the access protocol of the device, the virtual control steps of the device, the use information of the subnet layer and so on, such as SEDS1 and SEDS2. With the hierarchical structure of the system, the higher the upper layer, the higher the aggregation degree of SEDS and the less the number of SEDS. For example, seds5 integrates the relevant information of seds1 and seds2, and then adds the access interface of SPP (Space packet protocol). SEDS mainly describes the data exchange interface between services, including the input or output parameters, commands, business primitives, the relationship among them and the state machine representing the relationship between services (Fig. 2).

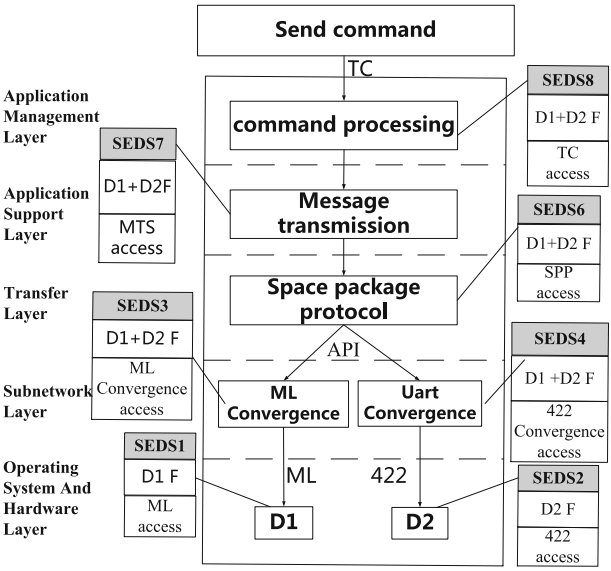

Fig. 2. SEDS describes the device, service interface

4 SEDS Implementation

In the CAST software architecture, 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 Fig. 3. 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.

4.1 Command 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

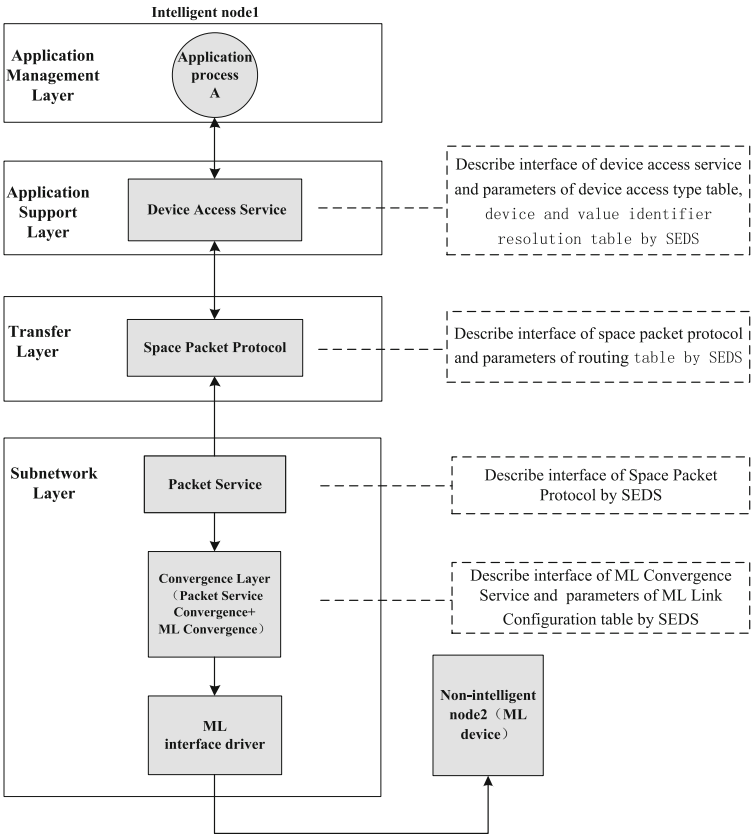

Fig. 3. ML command sending

by searching the device access type table (Table 1) by device id. Then, through searching device and value identifier resolution table (Table 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 1. Device access type table

Name	Device id (2Byte)	Corresponding device access type DAP (2Byte)
ML interface 1	0x8	24 (Sending data to the device DAP)
ML interface 2	0x9	24 (Sending data to the device DAP)

Table 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 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 4. ML link configuration information

Link	Link id (2Byte)	Link type (2Byte)	Driver Master (4Byte)	Driver Slave (4Byte)
ML1	0x6	0	3	1
ML2	0x7	0	3	2

4.2 Interface and Parameter Realization

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

1) Parameter configuration
   The parameter configurations to be described are shown in Tables 1, 2, 3 and 4.

```
<ContainerDataType name="device_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_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="dv_apid" type="uint16_t"/>
    <FixedValueEntry fixedValue="0" name="start_adr" type="uint32_t"/>
    <FixedValueEntry fixedValue="1000" name="dv_length" type="uint16_tl"/>
  </EntryList>
</ContainerDataType>

<ContainerDataType name="routing_table">
  <EntryList>
    <FixedValueEntry fixedValue="0x420" name="ro_apid" type="uint16_t"/>
    <FixedValueEntry fixedValue="0x7e0" name="ro_mask" type="uint16_t"/>
    <FixedValueEntry fixedValue="0" name="link_id" type="uint16_t"/>
    <FixedValueEntry fixedValue="0" name="next_link_id" type="uint16_t"/>
    <FixedValueEntry fixedValue="0"  name="s_routing_parameter" type="uint32_t"/>
  </EntryList>
</ContainerDataType>
<ContainerDataType name="routing_table">
  <EntryList>
    <FixedValueEntry fixedValue="0x7" name="ro_apid" type="uint16_t"/>
    <FixedValueEntry fixedValue="0x7ff" name="ro_mask" type="uint16_t"/>
    <FixedValueEntry fixedValue="6" name="link_id" type="uint16_t"/>
    <FixedValueEntry fixedValue="0" name="next_link_id" type="uint16_t"/>
    <FixedValueEntry fixedValue="0" name="s_routing_parameter" type="uint32_t"/>
  </EntryList>
</ContainerDataType>
```

2) interface

a) Device access services of the application support layer

External interfaces required:

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

b) Spatial packet protocol of the transport layer

Externally provided interfaces:

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

External interfaces required:

```
status_t (*snPacketSend_funcp) (uint8_t qos, uint8_t priority, uint8_t
channel, uint8_t
        next_link, uint8_t next_link_address, uint8_t
    *packet_buffer_p,uint32_t length)
```

c)Package service at the sub-network layer

Externally provided interfaces:

```
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)
```

Other interfaces required:

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

d) ML aggregation service of the data link layer

Externally provided interfaces:

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

```
<!-- 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="config" readOnly="true" type="uint32_t" mode="async" />
   <Parameter name="qos" readOnly="true" type="uint32_t" mode="async" />
  </ParameterSet>
 </Interface>
</DeclaredInterfaceSet>
<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="config" readOnly="true" type="uint32_t" mode="async" />
   <Parameter name="qos" readOnly="true" type="uint32_t" mode="async" />
  </ParameterSet>
 </Interface>
</DeclaredInterfaceSet>
<DeclaredInterfaceSet>
 <Interface name="snPacketSend_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" readOnly="true" type="uint8_t" mode="async" />
   <Parameter name="next_link_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>
```

4.3 Verification

Based on 43 software components of layered software architecture, CCSDS spatial link protocol, spatial domain protocol, 9 standard services and protocols of SOIS, 12 services and aggregation protocols of pus protocol of ECSS, and 46 service interface SEDS are developed. Seventeen SEDS are developed for the equipment connected with CTU, which are used as the input of unit test, assembly test and verification test. In the current software development process, saving about 30% of the time, automatic code generation accounted for 46%. At the same time, it solves the inconsistency of interface state caused by most file problems.

5 Concludes

In the layered architecture of spacecraft, SEDS can be used to describe equipment information and service interface information, which is suitable for all stages of spacecraft software development. SEDS can automatically generate software code, remote control instructions, telemetry analysis, test cases and data interface documents through tools, so as to shorten the software development cycle, improve the efficiency

of software development, and reduce the uncertainty and inconsistency of documents by automatically generating documents. Using SEDS to describe the data system is helpful to the rapid integration and testing of software, and its reusability and reliability are helpful to improve the development efficiency, so as to shorten the development cycle of the whole spacecraft.

Acknowledgement. This work is supported by the 'The National Key Research and Development Program' of China (No. 2018YFB1800301).

References

1. CCSDS 876.1-R-1 Spacecraft Onboard Interface Service – Specification for dictionary of terms for electronic data sheets for onboard components CCSDS (2016)
2. CCSDS 876.0-R-1 Spacecraft Onboard Interface Services – XML specification for electronic data sheets for onboard devices CCSDS (2016)
3. CCSDS.232.0-B-3 TC Space Data Link Protocol. CCSDS, Washington (2015)
4. CCSDS.732.0-B-3 AOS Space Data Link Protocol. CCSDS, Washington (2015)
5. CCSDS TBD.0-G-0 Electronic Data Sheets and Common Dictionary of Terms for Onboard Devices and Components
6. CCSDS. 871.2-M-1 Spacecraft Onboard Interface Services – Device Virtualization Service. CCSDS, Washington (2014)
7. Kawamura, M., Kawamura, M., Kawamura, M., et al.: Nonspacecraft onboard interface with spacecraft and spacecraft. J. Spacecr. Spacecr. **8**(2), 153–164 (2013)
8. Kawamura, M., Kawamura, M., Kawamura, M., et al.: Spacecraft onboard interface series-message transfer service. J. Spacecr. Spacecr. Interface **8**(1), 1–8 (2012)
9. CCSDS. 871.1-M-1 Spacecraft Onboard Interface Services – Device Data Pooling Service. CCSDS, Washington (20120
10. He, X., He, X.: Design of a spacecraft integrated electronic system service and protocol architecture. Spacecr. Eng. (2017)