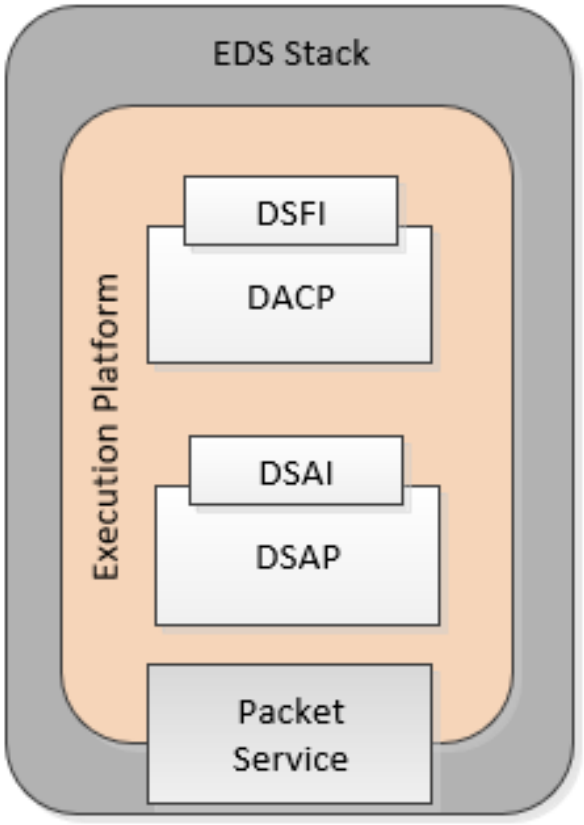
Doc. no: TER-OSRAD-RP-0005
Rev: 1
Date: 2022-01-07
DRD:

TERMA

Onboard Software Reference Architecture Demonstrator
Executive Summary Report



Prepared:

Poul Hougaard, Nicholas Mecredy, Dennis Bo Hansen

Senior Analyst, Senior Engineer, Systems Engineer

Authorized:

Digitally signed by Robert Olesen
Date: 2022.01.06 14:13:24 +01'00'

Robert Olesen

Product Assurance

Approved:

2022.01.06 13:53:17 +01'00'

Ole Hartnack

Project Manager

Record of Changes

ECR/ECO	Description	Rev	Date
-	Released	1	2022-01-07

Contents

Abbreviations and Acronyms

Abbreviation	Meaning
ACN	ASN Control Notation
ASN	Abstract Syntax Notation
DACP	Device Abstraction Control Protocol
DSAI	Device Specific Access Interface
DSAP	Device Specific Access Protocol
DSFI	Device Specific Functional Interface
EDS	Electronic Data Sheet
EU	Electrical Unit
FDIR	Fault Detection, Isolation and Recovery
GNDC	Ground Communication
HK	Housekeeping
ISO	International Standard Organization
OSRA	On-Board Software Reference Architecture
OSRAD	OSRA Demonstrator
PUS	Packet Utilization Standard
SAVOIR	Space Avionics Open Interface Architecture
SCM	Space Component Model
SECT	SAVOIR Electronic Data Sheets Common Tooling
SEDS	SOIS Electronic Data Sheet
SOIS	Spacecraft Onboard Interface Services
STR	Star Tracker
SW	Software
TASTE	The ASSERT Set of Tools for Engineering
TBD	To Be Defined
TC	Tele Command
TM	Telemetry

1 Introduction

This Executive Summary Report gives a short introduction to the project and summarizes the findings gathered during the OSRAD project activities. The summary is intended to be a self-contained description.

1.1 Project Objectives

The objective of the activity is to analyse how the on-board software reference architecture (OSRA) toolchain can be extended with a mechanism that support/allows (semi) automatic integration of instruments (devices), that has been specified by electronic data sheets (SEDS).

1.2 Project Background

1.2.1 Onboard Software Reference Architecture (OSRA)

The baseline for the project is the onboard software reference architecture. The architecture aims at 'separation of concern', meaning that mission independent functionality should be isolated in an 'execution platform', while mission dependent functionality belongs to the component layer (application layer).

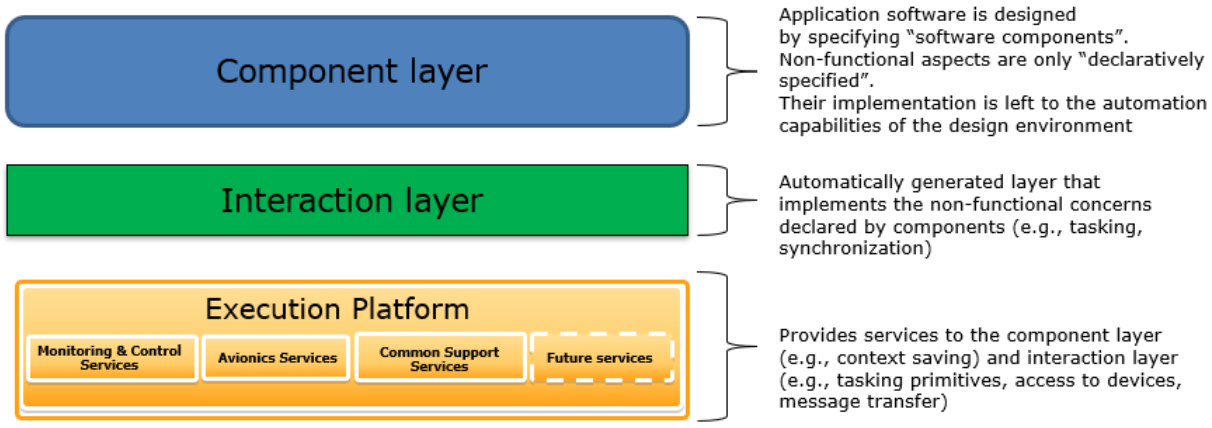


Figure 1-1: Overall Reference Architecture

1.2.1.1 Execution Platform

The execution layer provides all services needed for tasking, protection, synchronization, communication, etc. These services are used by the interaction layer. The services provided by the execution platform are independent of the application.

It is intended that the execution platform is reusable across missions. For this to succeed, the services integrated in the execution platform must be generic, e.g., if monitoring and control services are based on PUS, it shall be possible to configure the execution platform with the variability of PUS.

1.2.1.2 Pseudo Component

Some of the services provided by the execution platform are needed at component level. To expose execution platform services to the application, the concept of a *pseudo component* is introduced.

A pseudo component appears at the component layer as ordinary component (with provided services). However, a pseudo component has no implementation at component level but behaves as a kind of proxy for the execution platform services. The interaction layer is responsible for mapping component level requests to the corresponding execution platform services (to ensure the mission independence of the execution platform and move mission dependency to the component/interaction layers).

A pseudo component, representing the functionality of a device, is called a *device component*.

1.2.2 Electronic Data Sheets (SEDS)

The electronic data sheets applied are defined as the SOIS Electronic Data Sheets (SEDS).

SEDS is intended to replace the traditional user manuals, specifications, and data sheets that accompany a device and are necessary to determine the operation of the device and how to communicate with it.

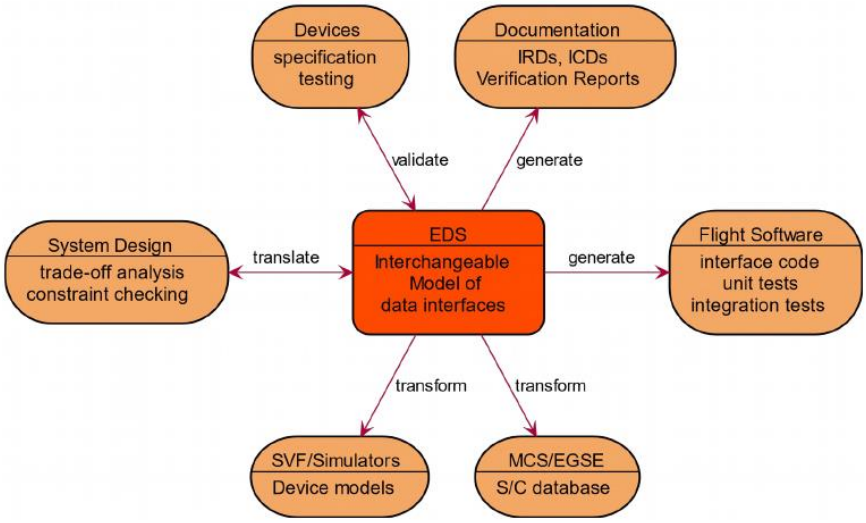


Figure 1-2: SOIS EDS Scope

An electronic data sheet is intended to describe a device interface in a formal way, allowing automatic processing of the specification.

Devices are mission independent, while the actual usage of a device will be mission dependent. This implies that the implementation of the device control belongs to the execution platform, while the actual usage belongs to the component layer.

1.3 Device

The example device used is the Terma T1 star tracker. The configuration consists of a head, connected to the 'electrical unit' (EU). The electrical unit is responsible for the communication with the onboard computer.

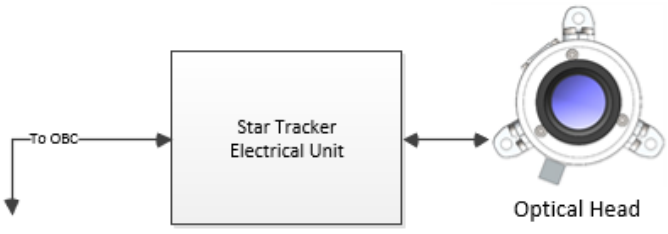


Figure 1-3: Star Tracker Configuration

2 Project Process

2.1 Specification of the Star Tracker SEDS

The star tracker SEDS is intended to describe the *interface* of the star tracker, as defined in the ICD. This implies that the resulting SEDS specifies how to monitor and control the star tracker but does not describe the semantics of the control. For example, the SEDS describes how to send a request for a mode change but has no knowledge of the expected resulting star tracker mode. Also, the SEDS is not able to prohibit illegal mode change requests.

The SECT tool supports the generation of interface descriptions in terms of html files. This is shown in Figure 2-1.

Figure 2-2 shows the corresponding section in the xml file.

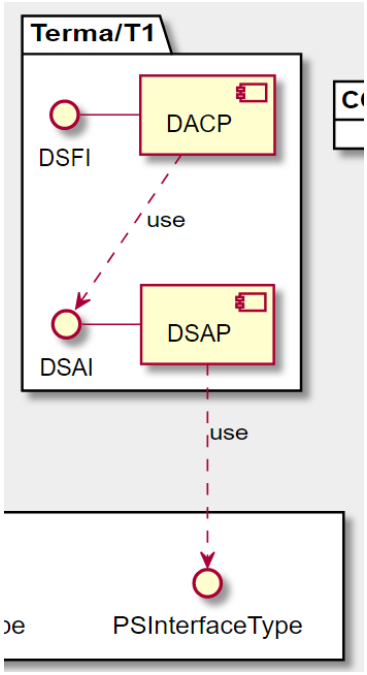


Figure 2-1: Generated ICD

```
<Component name="DSAP" shortDescription="Device-Specific Access Procedure">
    <LongDescription>...</LongDescription>
    <ProvidedInterfaceSet>
        <Interface name="dsai" type="DSAI" />
    </ProvidedInterfaceSet>
    <RequiredInterfaceSet>
        <Interface name="subnetworkPS" type="CCSDS/SOIS/Subnetwork/PSInterfaceType">
            <GenericTypeMapSet>...</GenericTypeMapSet>
        </Interface>
    </RequiredInterfaceSet>
</Component>
<Component name="DACP" shortDescription="Device Abstraction Control Procedure">
    <LongDescription>...</LongDescription>
    <ProvidedInterfaceSet>
        <Interface name="dsfi" type="DSFI" />
    </ProvidedInterfaceSet>
    <RequiredInterfaceSet>
        <Interface name="dsai" type="DSAI" />
    </RequiredInterfaceSet>
    <Implementation>...</Implementation>
```

Figure 2-2: Extract from Star Tracker SEDS

2.2 Integration of the device

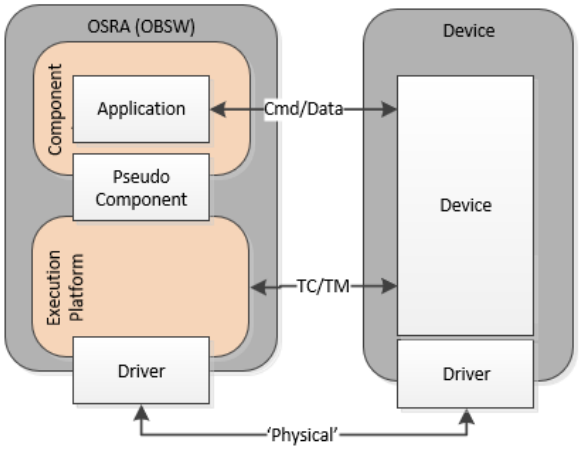


Figure 2-3: Baseline Configuration

Figure 2-3 illustrates the overall architecture with the on-board software represented by OSRA and a remote device, connected via a physical bus.

The *Execution Platform* is responsible for mission independent TC/TM management, the *Pseudo Component* is responsible for providing the services to the component layer, while the *Application* implements the mission specific process.

An SEDS specifies the device interface, and the integration results in functional code (TC/TM handling) in the execution platform and application interface specification in the component layer (Pseudo Component).

The star tracker SEDS defines components *DSAP* and *DACP*, taking care of different levels in the communication, as illustrated in Figure 2-4.

All SEDS defined functionality will be implemented in the execution platform. The monitoring and control services are provided at component layer by a Pseudo Component.

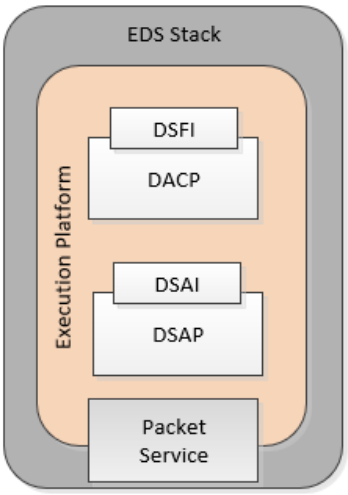


Figure 2-4: IO Stack Components

2.2.1 Device Specific Access Protocol (DSAP)

DSAP is defined as:

> The protocol that maps DACP to the applied SOIS subnetwork (Packet Service).
> DSAP is responsible for TC/TM management.

The DSAP provides the functionality in the *DSAI* interface.

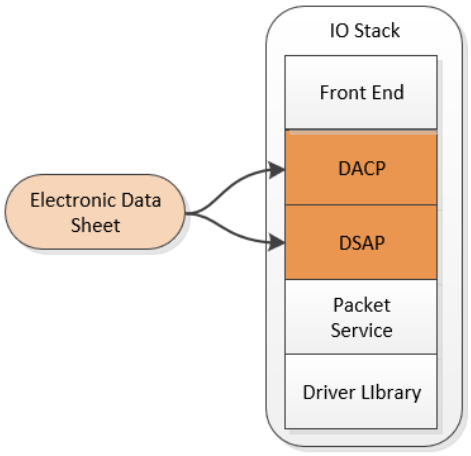
2.2.2 Device Abstraction Control Protocol (DACP)

DACP is defined as:

> The control procedure that provides the abstraction of a device-specific access protocol to a functional interface.

The DACP provides the functionality in the *DSFI* interface.

2.2.3 SEDS Integration in the Execution Platform



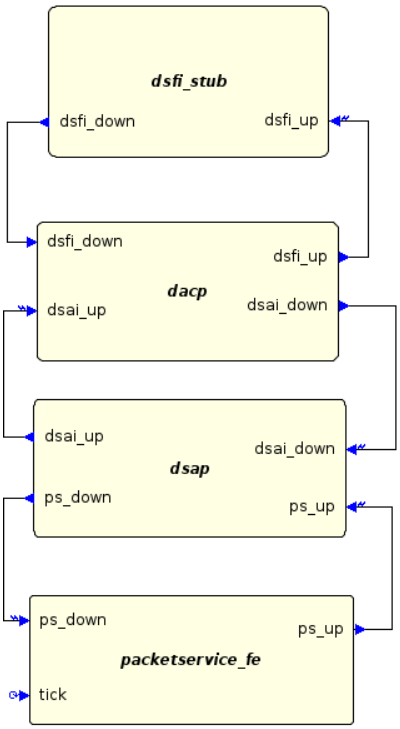
SEDS contains functional specification for mapping between interfaces and enables the generation of part of an IO stack, i.e., the DACP and DSAP components, as illustrated. The SEDS expects the existence of a 'Packet Service' layer, which in turn requests lower-level driver functionality.

The '*Front End*', '*Packet Service*' and *'Driver Library*' modules are not defined in the SEDS and must be hand coded.

Figure 2-5: SEDS Elements

For the integration of SEDS into the execution platform, it was decided to use SECT and TASTE tools:

Figure 2-6: Execution Platform generation



As shown in Figure 2-7, SECT/TASTE strategy is to provide/require one function and use the actual (TC)commands/(TM) packets as parameters. This simplifies the interface but leaves the actual identification of the commands/packets to the sequential code.

Figure 2-7: Generated IO Elements

2.2.4 SEDS Integration in the Component Layer

The pseudo component, representing the device, is constructed as outlined here.

The top-level interface (DSFI) is mapped to a pseudo component. The pseudo component's provided operations and data set emitters are generated, based on the interface's commands and parameters:

2.2.4.1 Commands

```
<Command      mode="async"
              name="STRModeCommand">
</Command>
```

This is mapped to an '`STRModeCommand` command.

2.2.4.2 Parameters

Parameters have several attributes:

```
<Parameter    mode="async"                          -- "async" or "sync"
              name="AttitudeInformation"
              readOnly="true"                       -- "true"/"false"
              type="ROUTINE_ATTITUDE_PARAMETERS"    -- defined in the DataTypeSet
              shortDescription="Routine attitude parameters"/>
```

An 'async' parameter will be provided as an 'data set emitter', i.e., will be updated automatically.

You can recognise this in the actual device component:

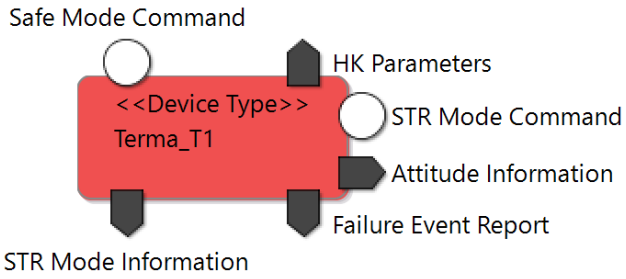


Figure 2-8: Device Pseudo Component

2.2.5 SEDS Integration in the Interaction Layer

The interaction layer is intended to be generated automatically from the application component model, based on the services provided by the execution platform. In principle, it is simply to call the related operations in the two layers. However, the introduction of data set emitting requires some data processing (the emit concept is not known by the execution platform). This is done by the introduction of a mapping module, illustrated in Figure 2-9 as the 'Bridge between STR Pseudo Component and DSFI'.

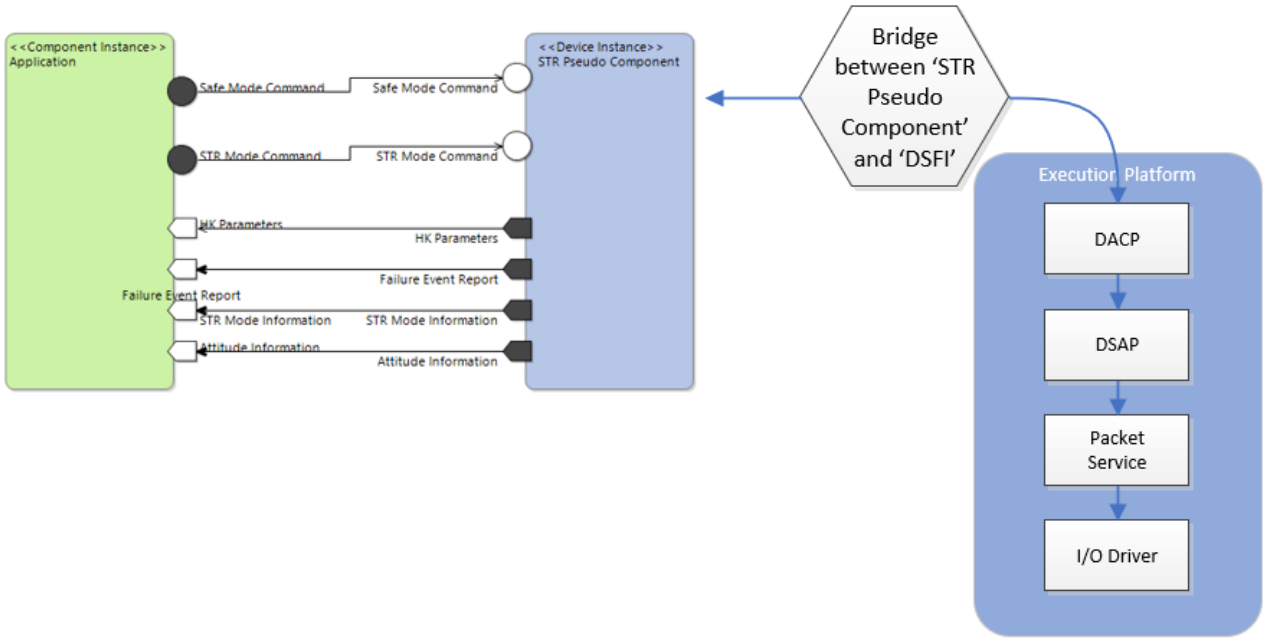


Figure 2-9: Interaction Layer Construct

2.3 Application Development

When developing the application, the execution platform and the corresponding interaction layer shall be seen as a black box, as illustrated below.

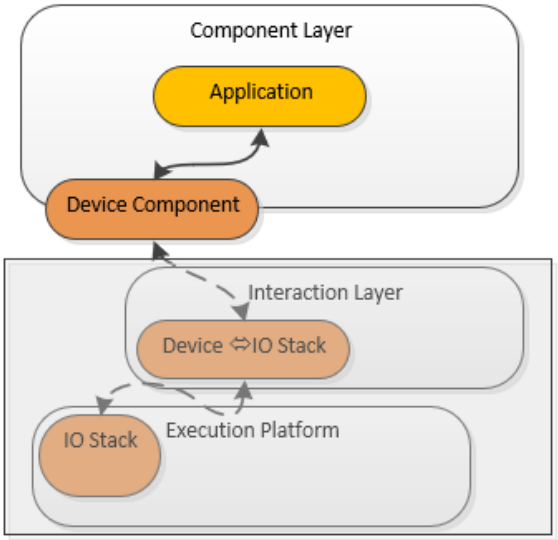


Figure 2-10: Application Development Environment

From the application point of view, the only visible result of the integration of the device is the Device (pseudo) Component.

Based on the System Requirements, an initial logical system model was produced as shown in Figure 2-11 with data as defined by the Data Dictionary found in Table 2-1

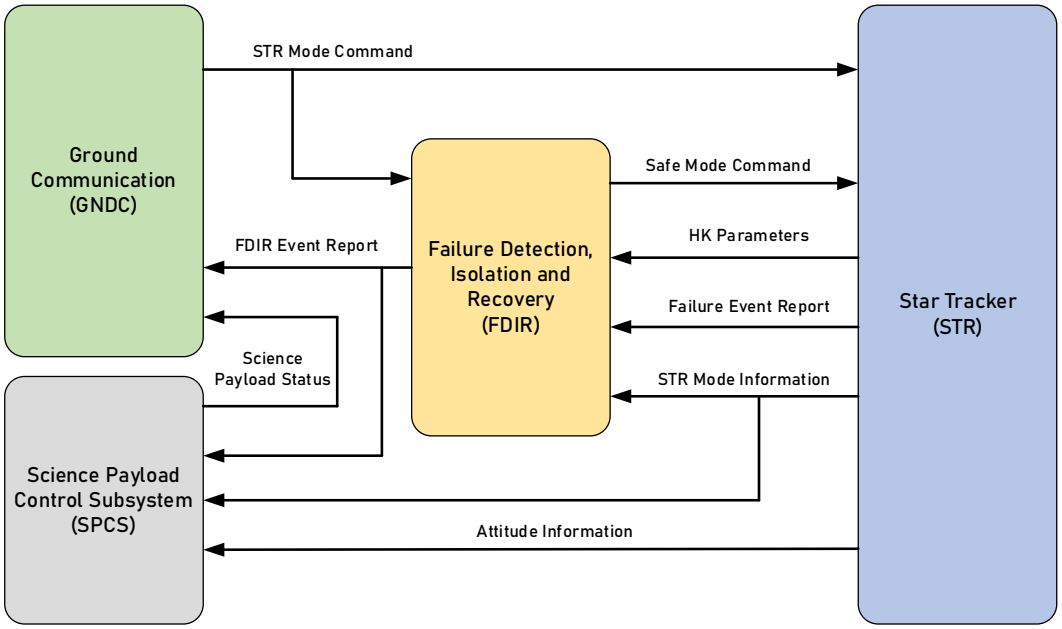


Figure 2-11: System Requirements Logical Model

Table 2-1: Data Dictionary	
Data Name	**Description**
STR Mode Command	Command that changes the mode in which the STR is currently operating.
Safe Mode Command	Command the STR into a safe mode
FDIR Event Report	Report that the FDIR has sent a Safe Mode Command to STR
Science Payload Status	Report about change in the Science Status parameter maintained by the SPCS.
HK Parameters	Parameters holding the values of selected STR housekeeping parameters.
Failure Event Report	Report that an event has occurred in the STR that may lead to reduced STR performance.
STR Mode Information	Information about the mode in which the STR is currently operating.
Attitude Information	The information that SPCS needs to verify STR attitude is *accurate* and *valid* and *current*. This information includes STR attitude, angular rate, and attitude validity information.

For specifying the software requirements, we use the logical model shown in Figure 2-12. This model is an elaborated version of the logical model used for the system requirements. The model shows the elements of the on-board application.

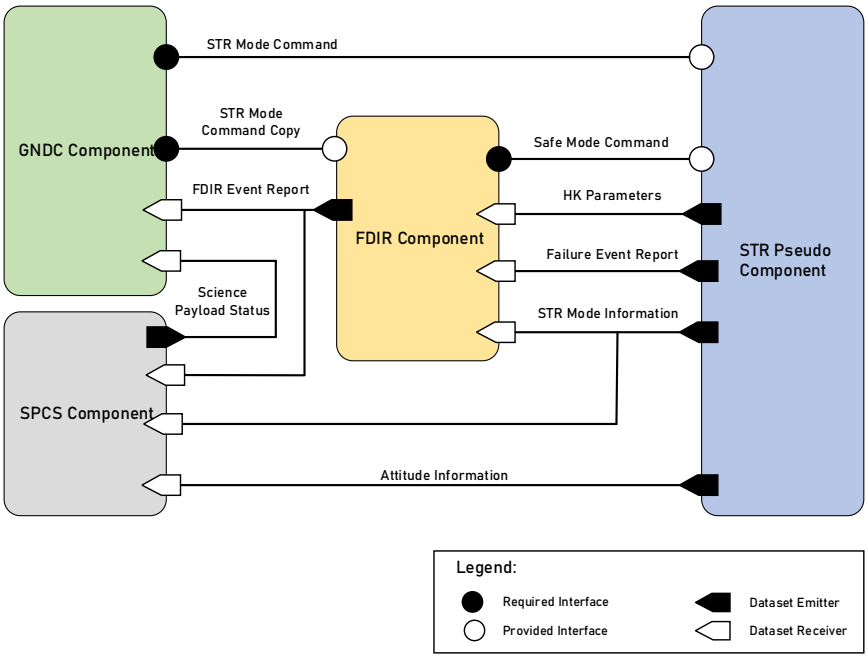


Figure 2-12 Software Requirements Logical Model

2.3.1 Application

The application initially registers as a data set receiver:

```
PseudoComponent_register_AttitudeInformation_DatasetReceiver(attitudeInformation_DatasetReceiver);
PseudoComponent_register_StrModeInformation_DatasetReceiver(strModeInformation_DatasetReceiver);
PseudoComponent_register_FailureEventReport_DatasetReceiver(anomalyReport_DatasetReceiver);
PseudoComponent_register_HKParameters_DatasetReceiver(housekeeping_DatasetReceiver);
PseudoComponent_register_ParameterValue_DatasetReceiver(parameterValue_DatasetReceiver);
PseudoComponent_register_InformationEventReport_DatasetReceiver(informationReport_DatasetReceiver);
PseudoComponent_register_command_verification_DatasetReceiver(application_commandVerification);
```

Figure 2-13: Dataset Receiver Registration

Then the application patiently waits until the upstart sequence has finished:

```
// upstartSequence: wait for autonomous change to atm - 1004
if (upstartSequence) {
    if (code == 1004) {
        upstartSequence = 0;
        printf("      Application: **** end of upstartSequence ****\n");
    }
 }
```

Figure 2-14: Wait for Upstart Conclusion

Subsequently, the application execution is controlled by the datasets received from the pseudo component, as illustrated in part of the output from the application.

```
Application started
Application: **** end of upstartSequence ****
Application: Got ROUTINE_ATTITUDE_PARAMETERS, time 26.001419 (10)
Application: Got ROUTINE_ATTITUDE_PARAMETERS, time 27.001419 (20)
Application: Got ROUTINE_ATTITUDE_PARAMETERS, time 28.001419 (30)
Application: Got ROUTINE_ATTITUDE_PARAMETERS, time 29.001419 (40)
Application: Got ROUTINE_ATTITUDE_PARAMETERS, time 30.001419 (50)
Application: Got ROUTINE_ATTITUDE_PARAMETERS, time 31.001419 (60)
Application: Got PARAM_HK_PARAMETERS (3)
Application: Got STR MODE 3, atm
Application: Command(1) SetParameterValue in ATM, HkPeriod = 5: Requested
Application: Command(1) SetParameterValue in ATM, HkPeriod = 5: Successful start of execution
Application: Command(1) SetParameterValue in ATM, HkPeriod = 5: Successful completion of execution
Application: Got ROUTINE_ATTITUDE_PARAMETERS, time 32.011734 (70)
```

Figure 2-15: Application Log

As shown, the application is informed about the status of the commands. This is managed by exporting the TM[1,*] verification packages as *command_verification_datasets*, which the application has registered as a receiver of.

3 Evaluation

3.1 Evaluation of SEDS

The main findings with respect to the SEDS are.

- It is not possible to define reactions to restriction violation. This implies that the reaction is tool dependent. We found that the applied tool chain provoked an execution crash when a restriction was violated.
- The SEDS is difficult to read and very laborious to write. This implies that it is error prone to write specifications. We did not have specialized editors available
- The SEDS standard specifies the syntax. The sematic is not defined and must be deduced based on the provided examples and from the actual syntax. This implies that specific semantics might first be clarified by the applied tool set.
- Validation of an SEDS. It is straight forward to validate the syntax towards the provided schema. Semantic validation is much more complicated.
  SECT can do some semantic checking (e.g., complains in case of missing references).
- It is not possible to define endianness on interface level. Endianness has to be defined on type definition level (so two definitions of the same type are needed).
- The SEDS is sequential and does thus not have means for protection against data overriding being caused by e.g., task switching.

3.2 Evaluation of Tool Chain

Detailed findings

- For arrays of dynamical length, the auto-generate tool makes the maximum possible size depending on the type of the length field. In the actual case resulting in a buffer size of 2,267,742,764,480 bytes.
- The information provided in the SEDS gives limits for interface purposes only. Application-level limits for use by FDIR must be specified elsewhere.
- Functions to convert TM/TC to/from internal representation is manual coded.
  In principle, the code could be autogenerated.

- Building an application with several devices is problematic, as we end up with multiple copies of the same common definitions
- The .pr files generated by SECT that should represent the state machines coming from SEDS are not accepted by TASTE
- 'Spare' fields in telecommands based on the same generic type must be named individually
- We found that in some case, the validation of a 32-bit unsigned integer failed, - the actual value was evaluated greater than $2^{32} - 1$.

3.3 Evaluation of SCM

Implementations of the SCM model in terms of model editors and code generations is still on the prototyping level. Consequently, it is difficult to export editors from one project to another.

However, using the SCM editor for application design, we found:

- Static registration as data set receiver. This means that the dynamic (state dependent) registration/un-registration as data set receiver is not possible. Consequently, data set filtering must be implemented in the functional code.
- Relations between provided and required operations are strictly one-to-one. This means that if there is a need for sending the same command to two components, you must duplicate the command, and send two commands. This implies a loss of semantics, which must be covered textually.
- During system and software requirement specification, definition of corresponding logical models in the SCM editor allowed for reuse/exchange of the models through the requirements phase to the component design phase.

4 Conclusion and Recommendations

SOIS EDS is intended for specification of devices. This implies that the SEDS functionality shall be mapped to the execution platform while the (top level) interface shall be mapped to a device pseudo component.

The SOIS EDS standard is intended to be rich enough to describe any device, and to allow for generation of all necessary device artefacts (ICD, user manual, tests, ..). This has implied that an SEDS specification is close to impossible to read and equally difficult to write.

The SEDS specification examples does not define the behaviour of the device, but only the extraction/packing of messages:

- They define the interfaces statically, and
- Define dynamic handling of incoming/outgoing messages.

Recommendations:

- Be very specific with the objective of the actual SEDS. This means that it must be clear if the SEDS shall define packing/unpacking of messages, if it shall be used to specify the behaviour of the device, or if it shall be used to define the ICD. The intention is of course that the SEDS could be used for everything. However, this requires tools for extracting and isolating the relevant parts of the SEDS.
- Authoring tools for writing and validation of SEDS specifications should be made available.
- Compilers for transferring SEDS to source code should be made available.
- Specification of the semantics of the SEDS definition is needed.