Leveraging the Eclipse Modeling Framework to work with Electronic Datasheets

L. Petersson[1], D. Perillo[2]
[1] Lukas Petterson, intern at ESA/ESTEC, Noordwijk, Netherlands
[2] Eng David Perillo PhD, ESA/ESTEC, Noordwijk, Netherlands
Email: lukas.petersson@esa.int, david.perillo@esa.int

Abstract — This abstract provides a practical guide to leverage the Eclipse Modeling Framework (EMF) for working with Electronic Datasheets (EDS). Starting from the SOIS EDS definition, available on the SANA website, it will be explained how to setup an EMF working environment, and how to generate a Tree Editor for editing and visualizing EDSs. It will also be explained how to exploit the Acceleo Model2Text (M2T) transformation language to navigate EDS models, and to generate artefacts in an almost automated manner. The problem of validating EDS models will also be discussed. A simple EDS use case will serve as a running example throughout the abstract. All the code mentioned in this abstract will be made available on the ESSR website.

1 Introduction

An Interface Control Document (ICD) is a specification document that describes the interface(s) to a system or subsystem, in terms of its inputs, outputs, and their relationships. When the ICD is provided as an unstructured, paper-based document, it can be hard for the user to retrieve information and translate them into artefacts (for instance, to produce code and test cases).

Electronic Data Sheets (EDS) is an alternative approach to paper based ICDs. It is an abstract concept of digitalized interface documentation proposed by the Consultative Committee for Space Data Systems (CCSDS). It is standardized in the context of the *Spacecraft Onboard Interface Services Area* (SOIS). The SOIS EDS standard is intended to be portable among different database implementations, and general enough to describe almost any device. This comes with the price that reading and editing EDSs models in their native interchange format (specifically XML) is quite tedious and error prone. Alternative approaches, such as using an User Interface or a Domain Specific Language, would therefore be preferred. Motivated by the need to support the increasing adoption of SEDS, and the consequent request for open-source tools in support to their utilization, in this abstract we will illustrate a practical way to leverage the Eclipse Modeling Framework (EMF) for working with SOIS EDSs models. This includes being able to edit and visualizing SEDS, to check their consistency, and to automatically translate them into code or documentation. An overview of the background material and adopted technologies will be provided in section 2. A practical workflow to setup the EMF workbench for EDS will be given in section 3. Code snippets for Model 2 Text generation will be provided in section 4 together with an example EDS representing a simple Device. The problem of checking EDS consistency models will be discussed in section 5. Our considerations will be finally provided in the Conclusions.

2 Background material

2.1 SOIS EDS

The SOIS EDS (SEDS) standard documents electronic devices of on-board spacecrafts with a semi-formal language. It provides modeling constructs to enclose communication and data handling information in an interchange format. The documentation of SEDS can be found in two documents named “Blue Book” and “Green Book” on CCSDS’s website. The SEDS standard is implicitly bound to XML as its underlying interchange format. Therefore, the Blue Book comes with an XML Schema Definition (XSD) file establishing the syntactic definitions of the EDS language [4]. The Blue Book defines the recommended standard of the XML specification of EDSs for on-board devices. It answers the “what” and “how” questions, by not only providing the language specification but also implementation guidelines of specific modeling patterns as XML snippets. The Green Book is an informal document with a wide range of information intended to assist readers in understanding SOIS documentation [5], such as the Dictionary of Terms (DoT) used in the standard.

Figure 1 provides an overview of the constructs available in the SOIS EDS standard, which are also implemented in the SOIS_EDS.XSD schema. Datasheet is the model root element, containing the Interface definition and a Package. As an alternative root to the Datasheet element, SEDS allows having models defined by a *Packagefile*, whose content *Package* can be referenced by *Datasheet* through the *XInclude* mechanism. One example of SEDS is the ccsds.sois.seds.xml *Packagefile* model [11], containing some *generally-useful types, including: standard c-like types with defined encoding and range timestamps, quaternions, spin rates and similar space-domain quantities, that* may be reused across in SEDS Datasheets.

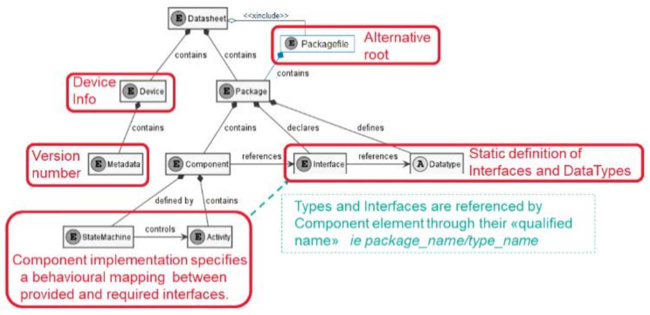

Datasheet
<<xinclude>>
Packagefile
Alternative root
contains
contains
contains
Device Info
Device
Package
contains
contains
declares
defines
Version number
Metadata
Component
references
Interface
references
Datatype
Static definition of Interfaces and DataTypes
defined by
contains
Types and Interfaces are referenced by Component element through their «qualified name» ie package_name/type_name
StateMachine
controls
Activity
Component implementation specifies a behavioural mapping between provided and required interfaces.

Figure 1 - Overview of Selected Key Elements and Abstract Types of a Datasheet

2.2 Eclipse Modeling Framework

EMF is a modeling framework and code generation facility used in Model Driven Engineering (MDE) for building tools and other applications based on a structured data model [9]. Such a model could for example be a SEDS instance, and it could be used to generate spacecraft engineering artefacts, such as elements for the System Database and flight software, test procedures and paper ICDs. All these options have the potential to facilitate and speed up the spacecraft integration.

The meta-model of an EMF project is defined using the Ecore language [10]. EMF generation technology can be used to automatically convert the EDS definition from its XSD format into the corresponding ecore metamodel, together with additional Java code [10]. The generated Java code can be used as it is, for instance to run an auto-generated Tree Editor for EDS; or it can be extended to implement additional capabilities, such as ad-hoc validators, custom editors or M2M and M2T transformations.

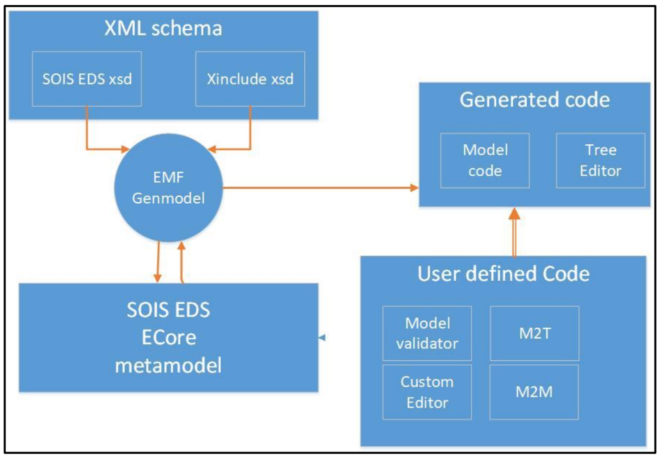
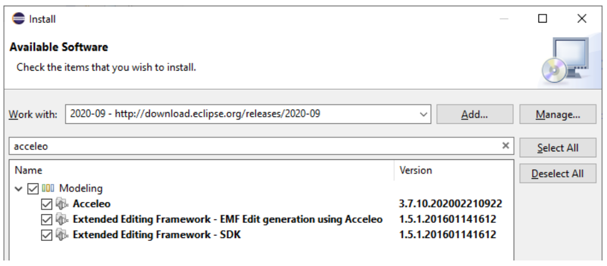


Figure 2 - EMF workflow starting from XML schema

3 Setting up of the EMF environment

To being able reproducing the code snippets and examples in this abstract, it is required to download and install the Eclipse Modeling Tools version 4.17.0; Java 14 and the Acceleo extension.



Install

Available Software

Check the items that you wish to install.

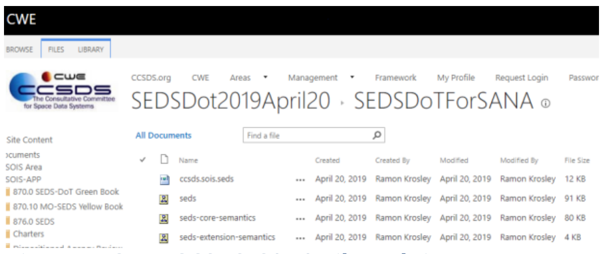
Work with: 2020-09 - http://download.eclipse.org/releases/2020-09 | Add... | Manage...

acceleo | Select All | Deselect All

Name	Version
☑ Modeling	
☑ **Acceleo**	**3.7.10.202002210922**
☑ **Extended Editing Framework - EMF Edit generation using Acceleo**	**1.5.1.201601141612**
☑ **Extended Editing Framework - SDK**	**1.5.1.201601141612**

Figure 3 - installation of additional eclipse plugins

The SOIS EDS schema can be downloaded from the SANA website [11]:



CWE

BROWSE FILES LIBRARY

CCSDS.org CWE Areas Management Framework My Profile Request Login Passwor

SEDSDot2019April20 › SEDSDoTForSANA

Site Content

ɔcuments

SOIS Area

SOIS-APP

870.0 SEDS-DoT Green Book

870.10 MO-SEDS Yellow Book

876.0 SEDS

Charters

All Documents Find a file

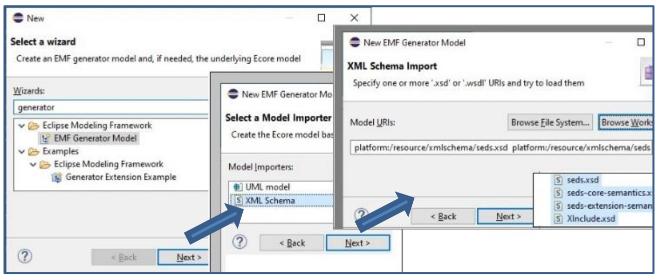
Name	Created	Created By	Modified	Modified By	File Size
ccsds.sois.seds	April 20, 2019	Ramon Krosley	April 20, 2019	Ramon Krosley	12 KB
seds	April 20, 2019	Ramon Krosley	April 20, 2019	Ramon Krosley	91 KB
seds-core-semantics	April 20, 2019	Ramon Krosley	April 20, 2019	Ramon Krosley	80 KB
seds-extension-semantics	April 20, 2019	Ramon Krosley	April 20, 2019	Ramon Krosley	4 KB

Figure 4 - CWE, CCSDS, SANA Files website

The XInclude schema can be downloaded from w3.org website [12].

3.1 Producing an ecore metamodel for SOIS EDS

Once all the above dependencies have been satisfied, and Eclipse has been restarted, it is possible to leverage the EMF Generator technology. Figure 5 depicts the basic steps to initialize an EMF Generator model (Genmodel), so as to automatically derive the ecore metamodel (seds.ecore) corresponding to the SOIS_EDS.xsd schema [11]. The xsd to ecore mapping implemented in EMF, also described in [13], translates almost all xsd constructs to equivalent ecore ones. However, the automatic translation has some known issues, such as the *non-repeating xsd:choice construct*. According to W3C, xsd:choice *"allows only one of the elements contained in the <choice> declaration to be present within the containing element"* [14]. The generated metamodel do not enforce the *non-repeating* nature of the constructs, but still contains the corresponding Entity and Relationships, with valid lower and upper bounds. The "*only one of the elements*" rule must be therefore manually implemented as Java code. In order to make the generated metamodel suitable to interpret SEDS models using XInclude mechanism, an Include Entity must be manually created. A useful feature when working with ecore is the possibility to enrich the metamodel with entity diagrams by simply dragging/dropping its element on a view. Figure 6 provides a View of the top-level elements in the SOIS EDS metamodel. The Include Entity has been created manually to support XInclude references. It matches the structure of includeType, as defined in the Include.xsd schema [12].



New

Select a wizard

Create an EMF generator model and, if needed, the underlying Ecore model

Wizards: generator

Eclipse Modeling Framework
- EMF Generator Model

Examples
- Eclipse Modeling Framework
  - Generator Extension Example

New EMF Generator Mo

Select a Model Importer

Create the Ecore model bas

Model Importers: UML model, XML Schema

New EMF Generator Model

XML Schema Import

Specify one or more '.xsd' or '.wsdl' URIs and try to load them

Model URIs: platform:/resource/xmlschema/seds.xsd platform:/resource/xmlschema/seds

Browse File System... Browse Works

seds.xsd, seds-core-semantics.x, seds-extension-seman, XInclude.xsd

< Back Next >

Figure 5 - initialisation of a Genmodel file

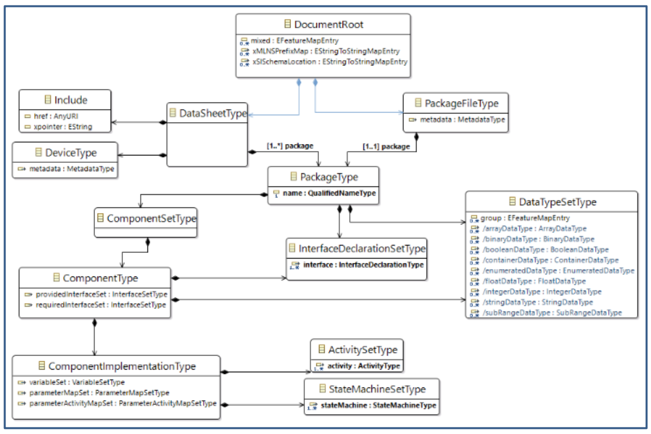

DocumentRoot
mixed : EFeatureMapEntry
xMLNSPrefixMap : EStringToStringMapEntry
xSISchemaLocation : EStringToStringMapEntry
Include
href : AnyURI
xpointer : EString
DataSheetType
PackageFileType
metadata : MetadataType
DeviceType
metadata : MetadataType
[1..*] package
[1..1] package
PackageType
name : QualifiedNameType
ComponentSetType
DataTypeSetType
group : EFeatureMapEntry
/arrayDataType : ArrayDataType
/binaryDataType : BinaryDataType
/booleanDataType : BooleanDataType
/containerDataType : ContainerDataType
/enumeratedDataType : EnumeratedDataType
/floatDataType : FloatDataType
/integerDataType : IntegerDataType
/stringDataType : StringDataType
/subRangeDataType : SubRangeDataType
InterfaceDeclarationSetType
interface : InterfaceDeclarationType
ComponentType
providedInterfaceSet : InterfaceSetType
requiredInterfaceSet : InterfaceSetType
ActivitySetType
activity : ActivityType
ComponentImplementationType
variableSet : VariableSetType
parameterMapSet : ParameterMapSetType
parameterActivityMapSet : ParameterActivityMapSetType
StateMachineSetType
stateMachine : StateMachineType

Figure 6 - top-level elements in the SOIS EDS metamodel.

Generating Tree Editor and descriptors

The .genmodel file enriches the Ecore model with properties required by the EMF generation engine. An exhaustive description of all Genmodel properties is out of scope for this abstract. Therefore, only the most relevant one for this application will be listed:

- *File Extensions:* xml (this is the default extension for SEDS)
- *Rich Client Platform:* true (to generate the editor with additional extension points, that can be used as standalone RCP application).
- *Model Directory (also Edit, Editor and Test Directory):* destination plugin name, followed by destination folder for code.
- *Model Plugin-ID*: desired name for the generated plugin

Figure 7 depicts the basic steps to generate a plugin project containing the Java code with the selected properties.

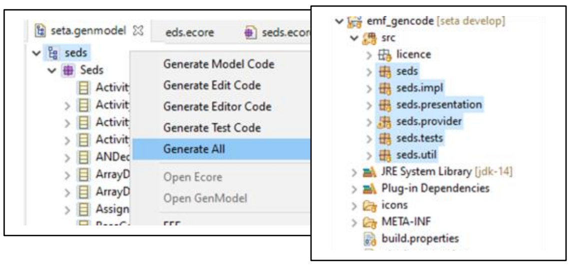
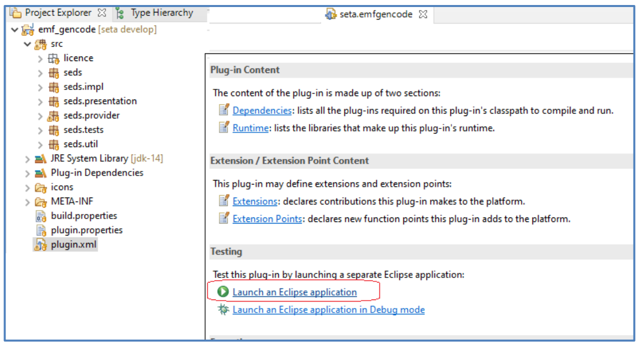


Figure 7 - emf code generation from genmodel

Executing the generated Tree Editor

The generated code contains a file named plugin.xml, from which the default Tree Editor can be executed as an Eclipse application (see Figure 8).
Figure 9 shows how an Electronic Datasheets looks like in the generated Editor. In particular, attributes of the selected model element can be edited in the Property table.



Project Explorer | Type Hierarchy | seta.emfgencode

emf_gencode [seta develop]
- src
  - licence
  - seds
  - seds.impl
  - seds.presentation
  - seds.provider
  - seds.tests
  - seds.util
- JRE System Library [jdk-14]
- Plug-in Dependencies
- icons
- META-INF
- build.properties
- plugin.properties
- plugin.xml

Plug-in Content

The content of the plug-in is made up of two sections:

Dependencies: lists all the plug-ins required on this plug-in's classpath to compile and run.

Runtime: lists the libraries that make up this plug-in's runtime.

Extension / Extension Point Content

This plug-in may define extensions and extension points:

Extensions: declares contributions this plug-in makes to the platform.

Extension Points: declares new function points this plug-in adds to the platform.

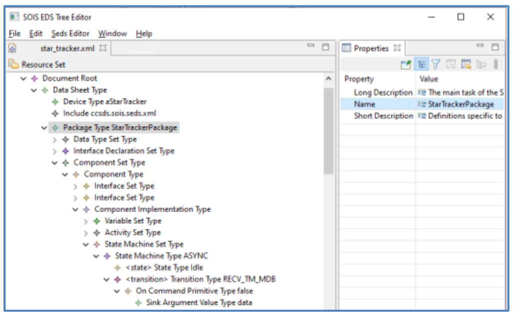
Testing

Test this plug-in by launching a separate Eclipse application:

Launch an Eclipse application

Launch an Eclipse application in Debug mode

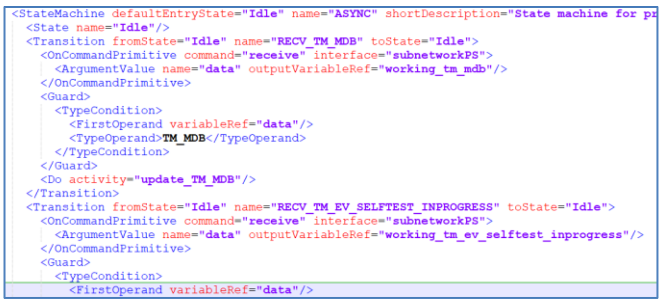
Figure 8 - execution of the generated tree editor



```
Resource Set
  Document Root
    Data Sheet Type
      Device Type aStarTracker
      Include ccsds.sois.seds.xml
      Package Type StarTrackerPackage
        Data Type Set Type
        Interface Declaration Set Type
        Component Set Type
          Component Type
            Interface Set Type
            Interface Set Type
            Component Implementation Type
              Variable Set Type
              Activity Set Type
              State Machine Set Type
                State Machine Type ASYNC
                  <state> State Type Idle
                  <transition> Transition Type RECV_TM_MDB
                    On Command Primitive Type false
                      Sink Argument Value Type data
```

Property	Value
Long Description	The main task of the S
Name	StarTrackerPackage
Short Description	Definitions specific to

Figure 9 - example SEDS visualized in the generated Tree Editor



```
<StateMachine defaultEntryState="Idle" name="ASYNC" shortDescription="State machine for pr
  <State name="Idle"/>
  <Transition fromState="Idle" name="RECV_TM_MDB" toState="Idle">
    <OnCommandPrimitive command="receive" interface="subnetworkPS">
      <ArgumentValue name="data" outputVariableRef="working_tm_mdb"/>
    </OnCommandPrimitive>
    <Guard>
      <TypeCondition>
        <FirstOperand variableRef="data"/>
        <TypeOperand>TM_MDB</TypeOperand>
      </TypeCondition>
    </Guard>
    <Do activity="update_TM_MDB"/>
  </Transition>
  <Transition fromState="Idle" name="RECV_TM_EV_SELFTEST_INPROGRESS" toState="Idle">
    <OnCommandPrimitive command="receive" interface="subnetworkPS">
      <ArgumentValue name="data" outputVariableRef="working_tm_ev_selftest_inprogress"/>
    </OnCommandPrimitive>
    <Guard>
      <TypeCondition>
        <FirstOperand variableRef="data"/>
```

Figure 10 - xml extract of the example SEDS

Integrating a M2T plugin to navigate and query the model

The Acceleo M2T transformation language is the Eclipse implementation of the MOFM2T standard, from The Object Management Group® (OMG®). It is delivered as an Eclipse plugin and provides means for generating textual artefacts from Ecore models. These can be either documents, reports, or code. Enabling a model-driven approach to building applications. This is, in many circumstance, way more convenient than writing all code from scratch [7]. Acceleo uses the Object Constraint Language (OCL), which is a formal specification language suitable to navigate and to define constraints on the elements of a model, based on the MOF [7]. Model navigation in OCL can be implemented with query expressions, which return information satisfying the stated constraints [6]. For example, the following expression can be used to retrieve instances of elements having a specific *Type*, within the scope of the node *PARENT*.

PARENT.eAllContents()->selectByType(*Type*)

Working with SEDS, the expression can be customized as to collect all State Machines contained in a Package (Figure 11, query collectStateMachine).

The Acceleo language is composed of two main types of structures, Templates and Queries. The former ones are the main modules for text generation. Templates can, in turns, execute Queries, which are constructs to process model elements with OCL [8]. A third kind of construct is the Java service, which is basically a mean to execute a Java method from within an Accelo Query. This can be particularly useful to use variable and complex branching condition which are not possible to implement with the Acceleo language.

4 Use Cases with result

The Use Case in this section provides an outlook of how to structure Acceleo code for generating a simple textual artefact: creating an Acceleo project in the workspace will produce 2 java files and one *.mtl* file. The *Activator* class controls the plug-in life cycle; whereas *Generate* class has the main java method to start the generation execution. It invokes the Acceleo generation methods with arguments defined in the .mtl file. Additional execution parameters, such as the input model and the output directory, must be specified in the launch configuration.
Once the generation method is executed, the user has full control of the generation flow as defined in the main Template structure of the mtl file.

The following Acceleo script navigates SEDS to generate a simple report of the State Machines contained in the Datasheet.

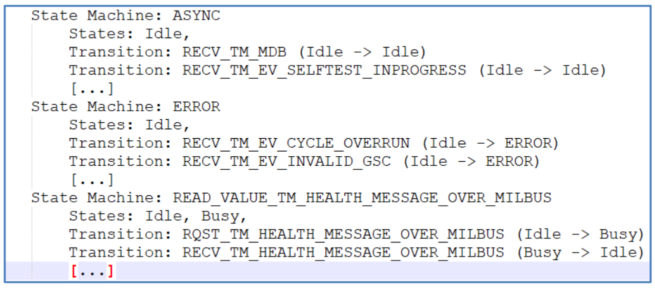
```
[comment encoding = UTF-8 /]
[module generate('http://www.ccsds.org/schema/sois/seds')]

[template public generateElement(aDataSheetType : DataSheetType )]
[comment @main/]:
[file ('Test', false, 'UTF-8')]
[for ( aSM : StateMachineType | aDataSheetType._package.collectStateMachines() )]
        State Machine: [aSM.name/]
                States: [for (aState : StateType | aSM.state)][aState.name/], [/for]
        [for (aTrans : TransitionType | aSM.transition)]
                Transition: [aTrans.name/] ([aTrans.fromState/] -> [aTrans.toState/])
        [/for]
[/for]
[/file]
[/template]

[query public collectStateMachines( p : PackageType) : Sequence(StateMachineType)
= p.eAllContents()->selectByType(StateMachineType)/]
```

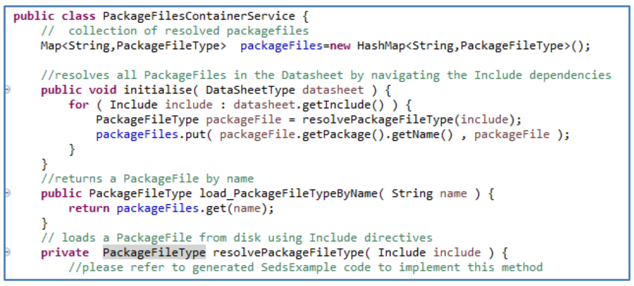
Figure 11 - Acceleo script reporting State Machines in SEDS

The imported metamodel (seds.ecore) is specified in the *[module generate(…)/]* directive (importing more than one metamodel is also possible). The name and the encoding of the output file are defined within the *[file(...)/]* directive. The main template of the transformation is identified by the *[comment @main/]* directive. All elements of type StateMachineType are collected in the *collectStateMachines* query. In the above script, main template has a for loop iterating on all element of type StateMachineType; it prints out the name of its States on one line, and their transitions below it.
The execution of the above script on the example SEDS file in Figure 10, produces the textual output in Figure 12. The indentation of the generated texts follows the indentation of the Acceleo script, making the output text easy to follow.
Even though Acceleo has no standard methods to resolve XInclude directives, it is still possible to implement an Acceleo service for this. In particular, this could be a Java class similar to the one in Figure 13. The PackageFiles collection is created by the initialize() method, loading all Include elements in the Datasheet and storing them into a Map.
PackageFiles can then be then accessed by their name using the load_PackageFileTypeByName(String). Method resolvePackageFileType can be implemented referring at the generated example code.



```
State Machine: ASYNC
    States: Idle,
    Transition: RECV_TM_MDB (Idle -> Idle)
    Transition: RECV_TM_EV_SELFTEST_INPROGRESS (Idle -> Idle)
    [...]
State Machine: ERROR
    States: Idle,
    Transition: RECV_TM_EV_CYCLE_OVERRUN (Idle -> ERROR)
    Transition: RECV_TM_EV_INVALID_GSC (Idle -> ERROR)
    [...]
State Machine: READ_VALUE_TM_HEALTH_MESSAGE_OVER_MILBUS
    States: Idle, Busy,
    Transition: RQST_TM_HEALTH_MESSAGE_OVER_MILBUS (Idle -> Busy)
    Transition: RECV_TM_HEALTH_MESSAGE_OVER_MILBUS (Busy -> Idle)
    [...]
```

Figure 12 - textual output of the Acceleo script executed on the example SEDS



```
public class PackageFilesContainerService {
    //  collection of resolved packagefiles
    Map<String,PackageFileType>  packageFiles=new HashMap<String,PackageFileType>();

    //resolves all PackageFiles in the Datasheet by navigating the Include dependencies
    public void initialise( DataSheetType datasheet ) {
        for ( Include include : datasheet.getInclude() ) {
            PackageFileType packageFile = resolvePackageFileType(include);
            packageFiles.put( packageFile.getPackage().getName() , packageFile );
        }
    }
    //returns a PackageFile by name
    public PackageFileType load_PackageFileTypeByName( String name ) {
        return packageFiles.get(name);
    }
    // loads a PackageFile from disk using Include directives
    private  PackageFileType resolvePackageFileType( Include include ) {
        //please refer to generated SedsExample code to implement this method
```

Figure 13 - example Java service class to make available Include dependencies to the Acceleo generation template

5 Considerations about EDS Validation

The validity of an EDS model shall be investigated in two directions: syntax and semantics. The first refers to the grammatical structure of the model, and the second to how the various symbols interrelate with each other. The generated EMF code performs a syntactic verification of the SEDS based on the Entities and Relationships of the seds.ecore metamodel. However, as already mentioned in section 3.1, the auto-generated seds metamodel do not enforce the *non-repeating* nature of the xsd:choice constructs, and a dedicated Java validation routine must be implemented to take care of this syntactic definition. As an alternative, SEDS files can be syntactically verified against the SEDS schema by any standard xml/xsd verification program, such as *"XML Notepad".*

On the other hand, the semantic validation of the SOIS EDS is way more difficult to implement. The SOIS EDS Blue Book contains many semantic rules, each leading to a potential semantic error. In the context of our analysis, we identified four semantic error categories:

Reference Errors and Type Consistency Errors. Are associated with broken references among elements in the model. Given that model elements in SEDS are referenced by their name, in plain text format, it is particularly common to miss-spell the name of the referenced element (Reference Error) or to reference an element having a type incompatible with the type of the destination argument (Type Consistency Error).

Literal value Errors. Some SEDS elements own an attribute to specify a default or initial value. This value is also just a String and its compatibility with the element's Type need to be checked. For instance, a Fixed Value of type Integer cannot be set to a non-integer value.

Name uniqueness Error. SEDS Elements within the same scope cannot have duplicate names.

Primitive Association Errors. SEDS defines synchronous and asynchronous primitives of interface types. Primitive Sink/Source elements which refers to a synchronous primitive must declare a Transaction attribute, which should be the same for all elements involved in the same transaction. A simplistic approach would be to verify whether the same transaction is used at least one Sink Primitive and at least one Source Primitive. A more thorough validation would imply to verify whether the Source and Sink primitives carrying the same Transaction value are connected by a Component.

The analysis of all validation rules and their implementation is probably out of scope for this abstract and it will be described in a future work.

6 Conclusions

To conclude, we have seen that SEDS are machine-readable interface specifications which are standardized under SOIS. The documentation and syntactic definitions of SEDS are available on the CCSDS website [11]. Those documents state that a SEDS has to include interface descriptions, protocol and procedure descriptions and documentation about the device. With this definition, a SEDS meta-model can be created using EMF. The meta-model can then be used to generate a number of spacecraft engineering artefacts such as elements for the System Database and flight-software, integration test procedures and paper ICDs. All these options have the potential to facilitate and speed up the spacecraft integration. In addition to this, we demonstrated how such a model can be transformed to a human-readable text file with Acceleo M2T, an Eclipse plugin which uses OCL to navigate models and generate a txt file. An advantage of this approach is that the same transformation can be used across different SEDS models, generating text files having the same format. We have also discussed the need for semantic validation of SEDS. A detailed identification of semantic errors and how to detect them in SEDS will be subject of a future work.

References

[1] En.wikipedia.org. n.d. Interface control document - Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/Interface_control_document [Accessed 24 June 2021].

[2] Taylor, C., 2014. [online] Indico.esa.int. Available at: https://indico.esa.int/event/57/contributions/2705/attachments/2244/2597/Adoptions_of_EDS_and_Device_Virtualisation_for_Onboard_Devices.pdf [Accessed 24 June 2021].

[4] Public.ccsds.org. 2020. Blue Books: Recommended Standards. [online] Available at: https://public.ccsds.org/Publications/BlueBooks.aspx [Accessed 24 June 2021].

[5] Public.ccsds.org. 2020. Green Books: Informational Reports. [online] Available at: https://public.ccsds.org/Publications/GreenBooks.aspx [Accessed 24 June 2021].

[6] Cabot, J. and Gogolla, M., 2012. Object Constraint Language (OCL): A Definitive Guide.

[7] En.wikipedia.org. n.d. Acceleo - Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/Acceleo [Accessed 24 June 2021].

[8] Di Natale, M., 2013. Code Generation (ModelToText o M2T) with Acceleo. [online] Available at: http://retis.sssup.it/~marco/files/lesson22-Acceleob.pdf [Accessed 24 June 2021].

[9] Gronback, R., n.d. Eclipse Modeling Project. [online] Eclipse.org. Available at: https://www.eclipse.org/modeling/emf/ [Accessed 24 June 2021].

[10] Vogel, L., 2019. Eclipse Modeling Framework (EMF) - Tutorial. [online] Vogella.com. Available at: https://www.vogella.com/tutorials/EclipseEMF/article.html [Accessed 24 June 2021].

[11] CWE, CCSDS, SANA Files https://cwe.ccsds.org/sois/docs/SOIS-APP/SANA%20Files [Accessed 24 June 2021]

[12] XInclude schema https://www.w3.org/2001/XInclude/#related.resources

[13] Eclipse.org. 2021. [online] Available at: https://www.eclipse.org/modeling/emf/docs/overviews/XMLSchemaToEcoreMapping.pdf [Accessed 24 August 2021].

[14] W3schools.com. 2021. XML Schema choice Element. [online] Available at: https://www.w3schools.com/xml/el_choice.asp [Accessed 24 August 2021].

[15] EMF https://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/codegen/ecore/genmodel/GenModel.html [Accessed 24 August 2021].

Author/Speaker Biographies

Lukas Petersson, Engineering Physics student at Lund University, has joined ESA as Intern at the TEC-SWF section. He is focusing on the Eclipse Modeling Framework and SOIS EDS.

David Perillo, Software Engineer with a PhD in emerging digital technologies. He joined ESA in 2020, after more than 10 years working in the Defence Industry. In TEC-SWF he is looking after R&D activities related to EDS technologies, MDE, software-emulators, and critical software for payloads and satellites.