The Consultative Committee for Space Data Systems

Technical Note Concerning Space Data System Standards

FUNCTIONAL RESOURCE REFERENCE MODELLING CONCEPTS

DRAFT AREA TECHNICAL NOTE

CSSA 3-TN

October 2021

DOCUMENT CONTROL

Document	Title and Issue	Date	Status
	1.1 Update for XSD generation	November 2021	

CONTENTS

Table Page

NO TABLE OF CONTENTS ENTRIES FOUND.

1 REFERENCES

[1] *Functional Resource Reference model*. Draft Area Technical Note, CSSA 001.0-TN-0.14, July 2018

[2] *Functional Resource Registry at SANA,* Technical Note, CSSA 2-TN-1.2, November 2018

2 INTRODUCTION

2.1 PURPOSE AND SCOPE

This document describes the workflow and tools used by the CSS Area to model the so called Functional Resource Model. The concepts and definitions of Functional Resources are described in [1]. Users of Functional Resources will access Functional Resources by means of the SANA registry[1]. The interface of the CSS area to SANA is covered by [2].

[1] https://beta.sanaregistry.org/r/functional_resources/functional_resources.html

3 FRM MODELLING WORKFLOW

Figure 1 shows the basic workflow of the FRM modelling. Members of the CSS area edit the FRM with an FRM editor, which produces the FRM model in an XML file suitable for import at SANA.

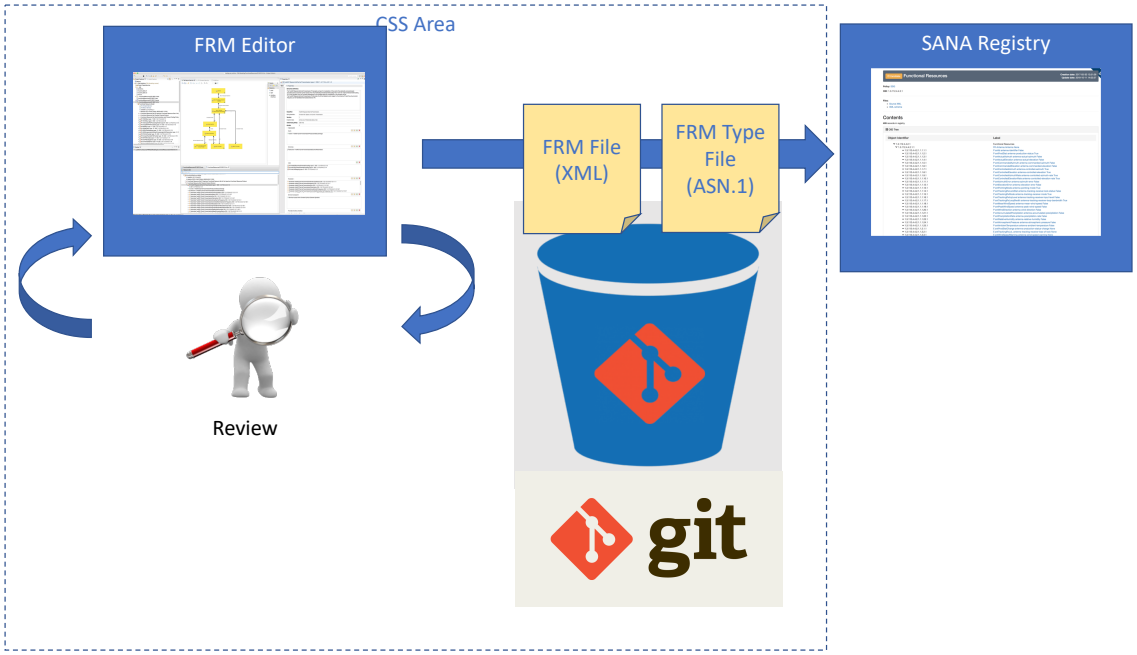


Figure 1 FRM Workflow

In the future it is foreseen to put the FRM files in a git repository hosted on the GitHub, this is discussed at CCSDS level.

The FRM editor supports the consolidation of individual FR files into a consistent FRM file allowing simple copy and past actions or drag and drop on level of FRs (and other levels if required).

From the FRM several products can be exported

- XML file for SANA import (actually the native FRM model serialization)
- HTML ready MS Word import for review
- ASN.1 for the type definitions of parameters, event values and directive qualifiers
- XSD for parameters marked as configurable

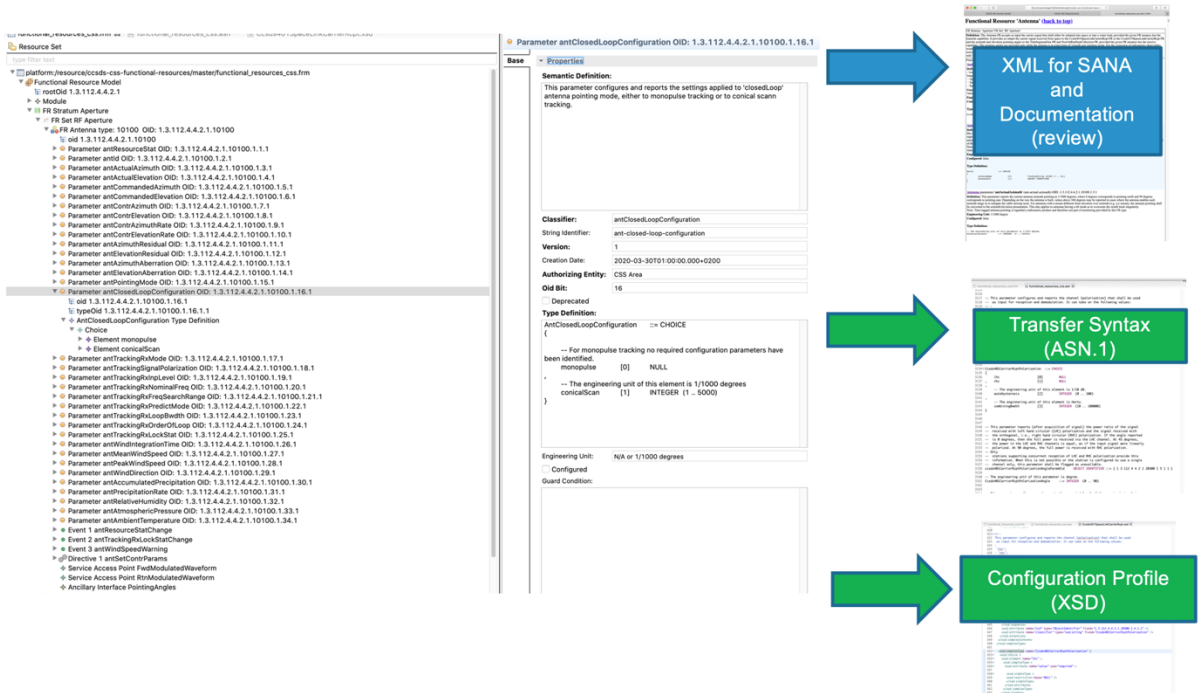


Figure 2 FRM Editor Product Export

4 FRM MODELLING FUNCTIONS

4.1 BASIC FRM FUNCTIONS

The FRM editor allows the creation, update and deletion of FRs by means of structured editor, supporting and enforcing the defined (XML) structure of the FRM as shown in Figure 3

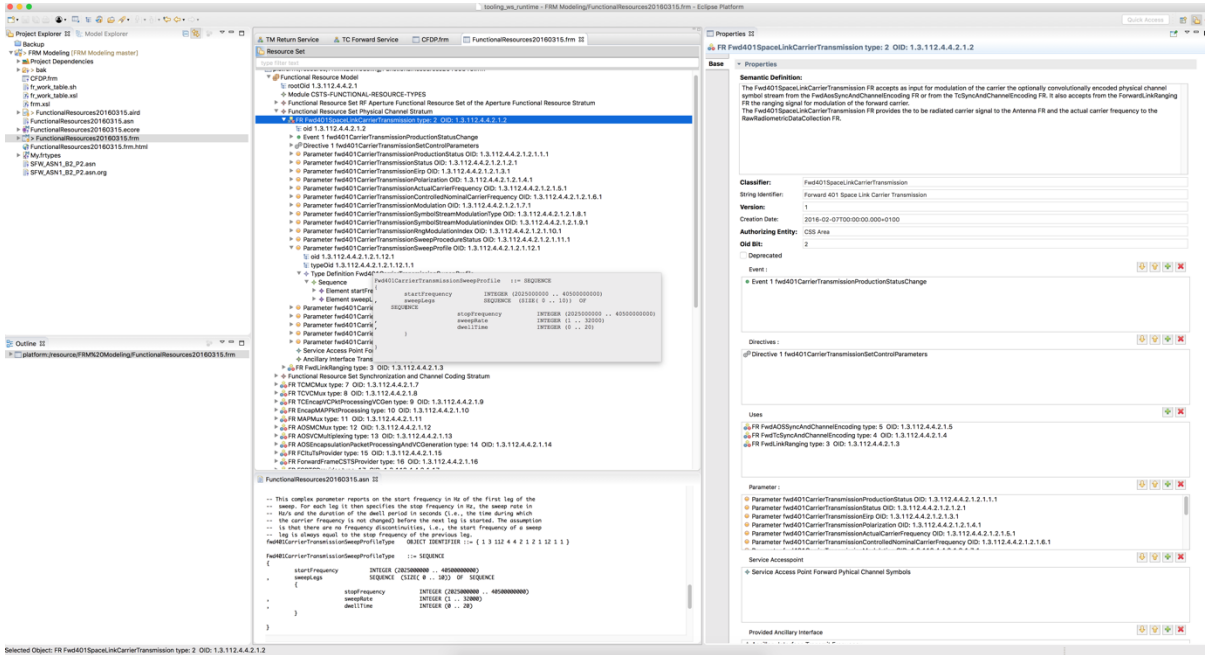


Figure 3 Functional Resource Editor

4.2 AUTOMATIC OID GENERATION

The FRM contains at the time of writing about ~750 OIDs subject to registration at SANA. Keeping this OIDs consistent is one of the key features of the FRM editor. The FR editor provides the means to automatically generate the OIDs of

- FRs
- Parameters
- Event
- Event Values
- Directives
- Directive Qualifier

The OIDs are generated using ‘offsets’ as specified per Functional Resource Set. In that way new Functional Resources can be added to Functional Resource Sets without the need to use ‘ the next free Functional Resource OID’, but the next free Functional Resource OID within the Functional Resource Set.

In addition the corresponding type OIDs, identifying the ASN.1 types in the generated ASN.1 module, are generated.

The automatic OID generation is invoked by using toolbar button as shown in Figure 4.

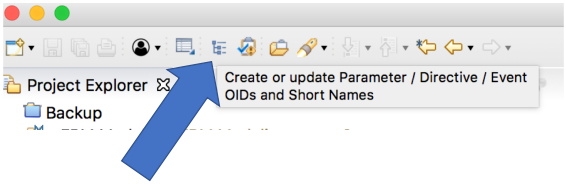


Figure 4 Automatic OID Generation

The logic of the OID generation takes the order of FR, P/ E / D / EV / DQ as they appear in the tree (file) and creates OIDs with a top down numbering. The general structure of Functional Resource OIDs are shown in Figure 5.

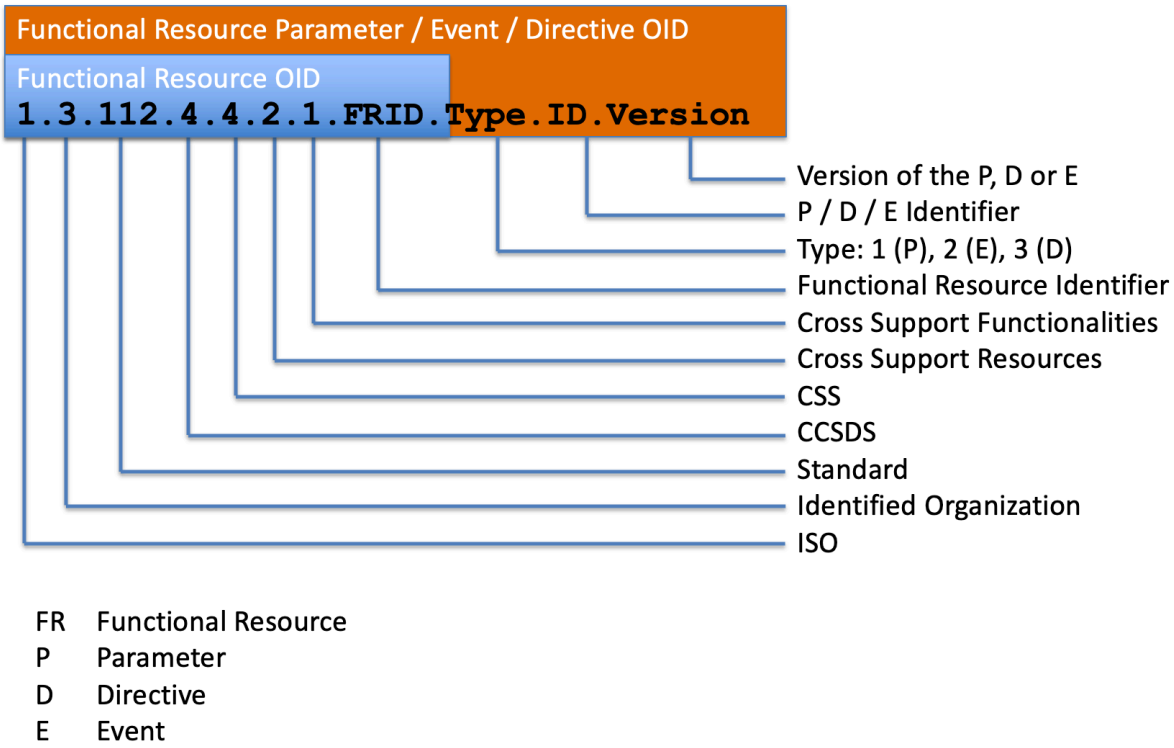


Figure 5 Functional Resource OID Structure

This approach is required for the initial preparation of the FRM as it allows FR (P/E/D) reshuffling and then consistent OID generation. However, subsequent invocations of the OID generation will preserve existing OIDs and will create new OIDs as needed. If a P/E/D with the same ‘classifier’ following each other are found, only the ‘version’ portion of the OID is increased. That allows creation of different versions of the parameters, events and directives over time.

Once the FRM registry has been published at SANA, OIDs must not be changed anymore, only new OIDs can be added.

4.3 TYPE DEFINITION AND ASN.1 TYPE MODULE GENERATION

The CSS/CSTS working group has agreed that the FRM shall provide type definitions for the types defined in the FRM. The basic rationale is to foster type compatibility among different implementations of CSTS Monitored Data Service and the upcoming CSTS Service Control. A dedicated ASN.1 module captures the FRM types and is published along with the FRM itself at SANA.

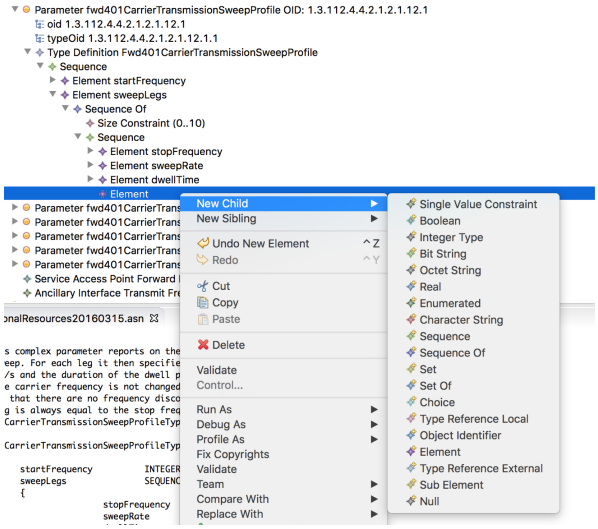


Figure 6 Define Types in FRM Editor

To support the generation of an ASN.1 type module from the FRM a Type Specification Language has been built into the FRM editor, see section 4.7; basically the FRM Type Specification Language is realizes a subset of the ASN.1. For use in the FRM editor this approach maps the ASN.1 types into the hierarchical structure of the FRM editor and allows to specify the required types. To support the person editing an FR, an 'in place preview' of the generated ASN.1 is provided as a tooltip.

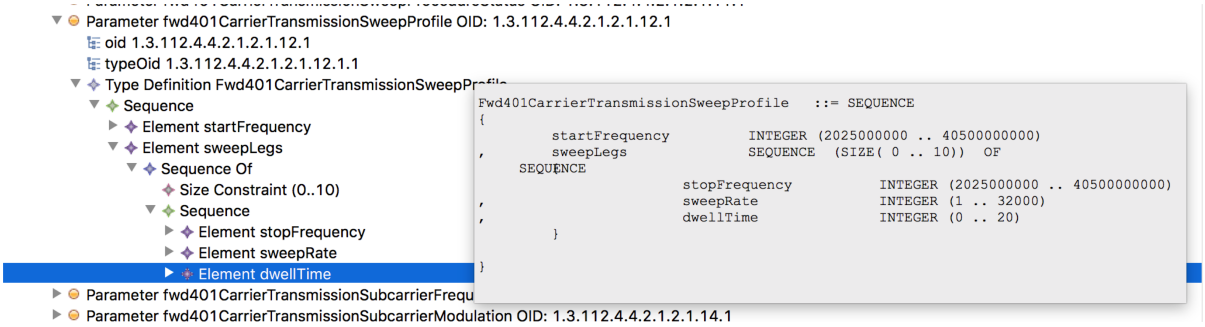


Figure 7 ASN.1 In place Preview

When generating the ASN.1 module, all generated types are gathered and the required export statements are generated. This allows users of the FRM ASN.1 file to directly feed the FRM ASN.1 file into an ASN.1 compiler and to use the generated artefacts for development and maintenance of CSTS services.

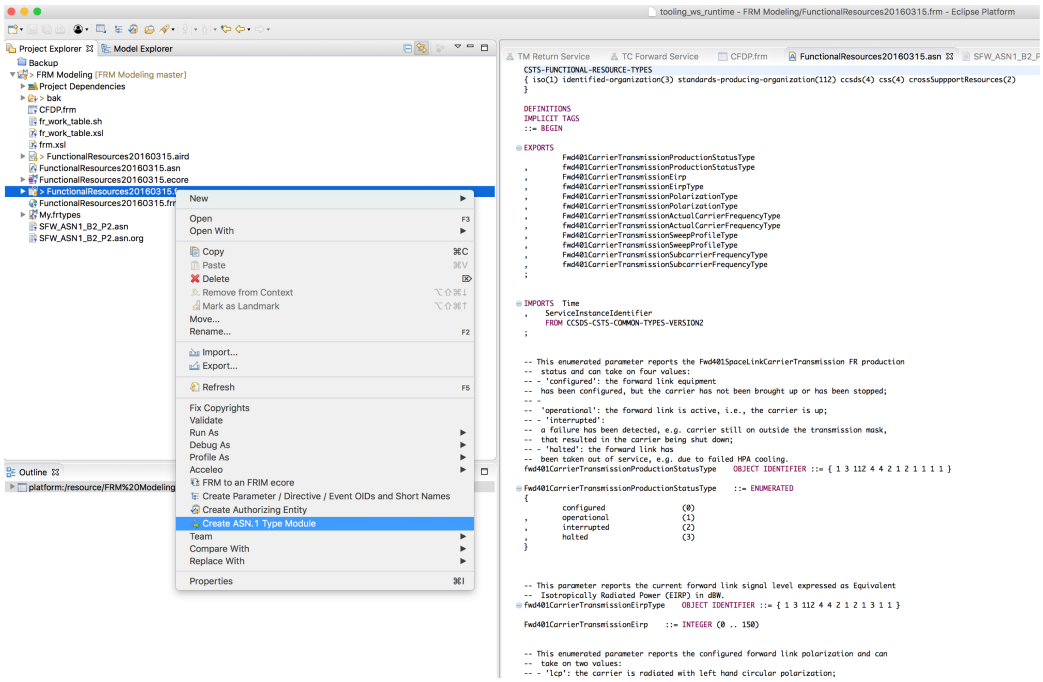


Figure 8 FRM ASN.1 Type Module

4.4 TYPE DEFINIONS AND XSD GENERATION

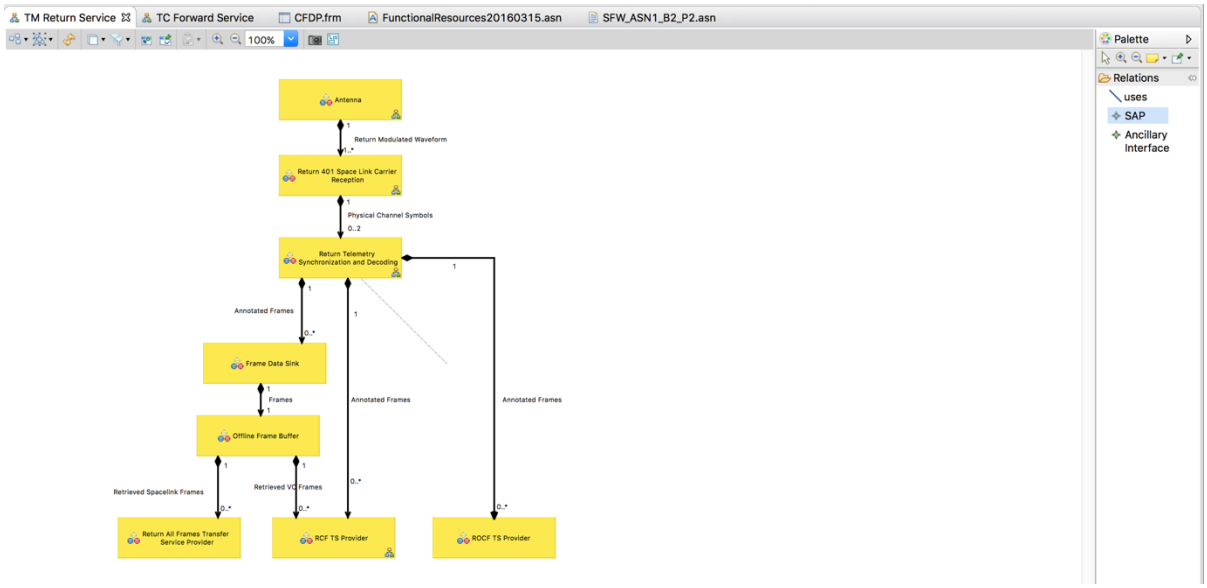
To support the definition of Configuration Profiles as done by the CSS/CSSM WG, the FRM editor allows the export of parameters flagged as ‘configurable’ as XSDs. This exploits the fact, that very often configuration parameters are also subject to monitoring and thus are defined as part of the FRM.

4.5 FUNCTIONAL RESOURCE SETS AND SERVICE ACCESS POINTS

Reference [4] uses so called Functional Resource Sets to group FRs. Furthermore FRs can expose Service Access Points (SAP) and Ancillary Interfaces to model relations among FRs. These two features are supported by the FRM editor, the latter in a graphical way to ease the modelling of FR relations.

4.6 GRAPHICAL REPRESENTATION

The FRM editor support graphical modelling of FR relations by specifying SAPs for FRs.



To use the graphical editing follow the steps below in the FRM Editor.

1. Switch to the Modeling Perspective: Window -> Open Perspective -> Modeling
2. If your project from 5) is not a modelling project convert it to be a Modeling project: right click the project -> Configure -> Convert to Modeling Project
3. Right click the project -> View Point Selection -> Functional Resources. Your project has now an representation.aird file
4. Right click the project -> Create representation. Select Functional Resource Diagram -> Next
5. Select the package from the FRM file for which you want to create a diagram
6. Populate the diagram by dragging and dropping element from the FRM file on the diagram

4.7 FRM TYPE DEFINION LANGUAGE

The FRM Type Definition Language has been modelled to provide a subset of ASN.1 for the definition of FRM types. While the prime intention is to generate ASN.1 out of these type definitions, a secondary goal is to generate a corresponding XML Schema (XSD to support the definition of CSSM data types in the context of Configuration Profiles. Figure 9 shows the structure of the FRM Type Definition language.

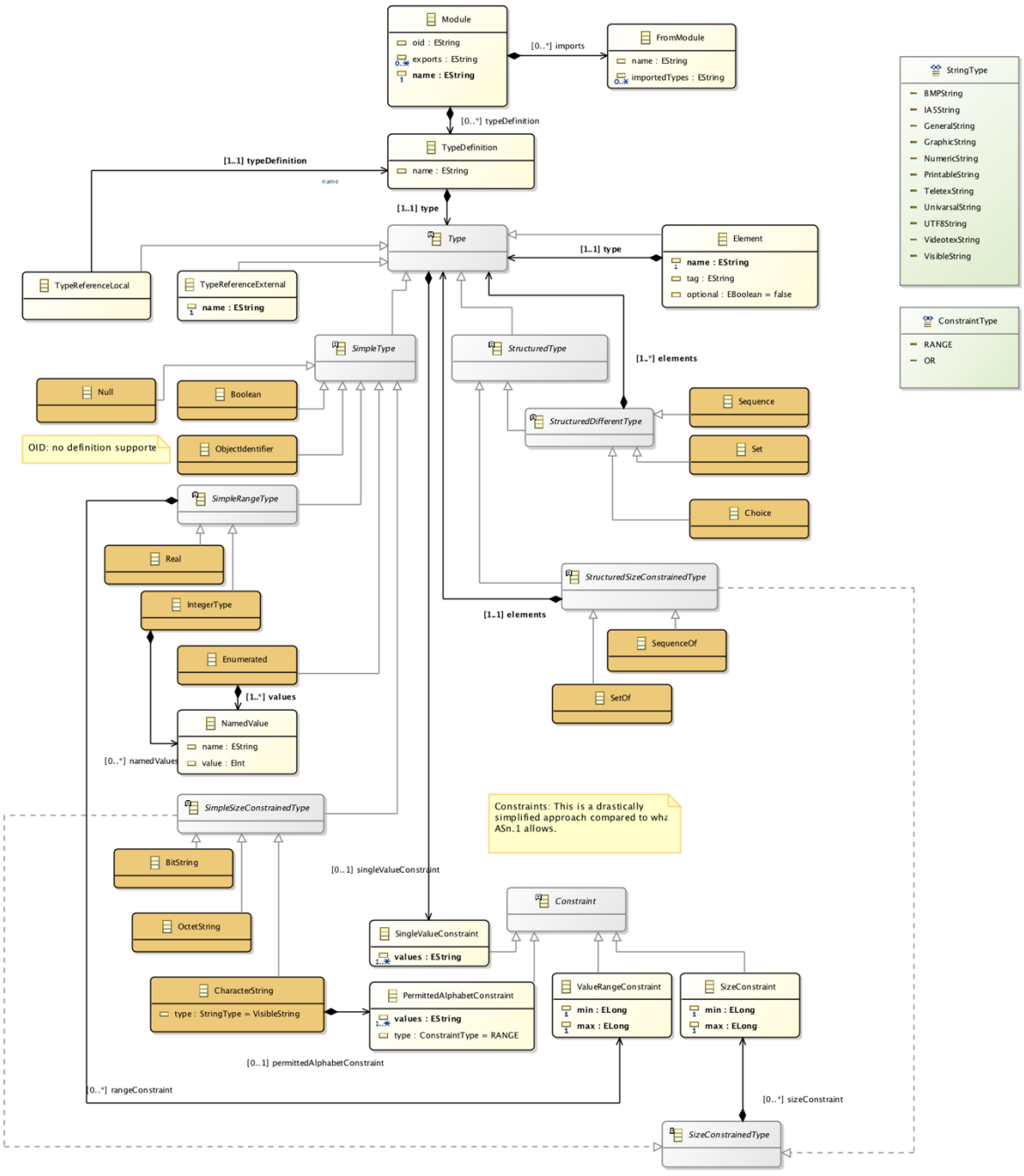


Figure 9 FRM Type Definition Language Model

4.8 FUNCTIONAL RESOURCE INSTANCE MODEL

This is an experimental feature allowing the transformation of the FRM model into a Functional Resource Instance Model (ecore). The underlying motivation is that at some point also the FR Instances (FRI) need to be modelled and used. At the moment this feature does not have any practical use, however there is a clear potential to extent the model driven approach to further use cases.

The basic idea is to create for each FR *objec*t an FRI *class*, supporting FRI instantiation to allow modelling of arrangement of FRIs. The basic idea is depicted by Figure 10.

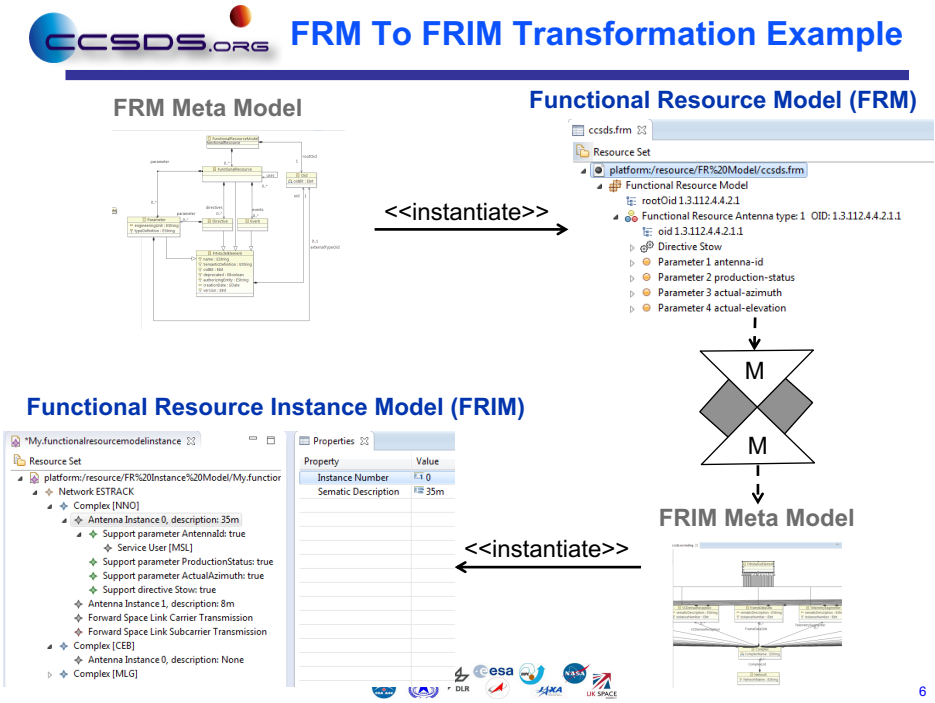


Figure 10 FR to FRI Model Transformation

Note: The generated FRI ecore model can be used in UML tools, which may be interesting. In an early stage of the Sequence of Events conception, the FRI classes (antenna etc.) have been considered in a conceptual model of the Sequence of Events.

5 FRM MODEL EDITOR ARCHITECTURE – FOR FRM EDITOR DEVELOPERS

5.1 GENERAL APPROACH

The FRM editor relies on the Eclipse Modelling Framework (EMF) to reduce the required development effort of the FRM Editor to the minimum. In a nutshell EMF allows the

definition of datatypes in a UML class diagram like style (ecore), which has been done for the FRM metamodel and the FRM type metamodel.

Based on these ecore type definitions EMF generates a structured eclipse (or RCP) editor for the FRM. The FRM itself is saved as XML (optionally XMI). In addition EMF supports the generation of an XML Schema (XSD) for the produced XML files.

5.2 EEF PROPERTIES

To edit the FRM, the simple standard EMF property pages generated are not suitable. However, the EEF project provides means to generate for ecore models (like FRM) more sophisticated form based property pages supporting the FRM editing. EEF property pages have been used for the FRM editor.

The steps outlined by

https://wiki.eclipse.org/EEF/Tutorials/First_Generation

have been applied to the generated EMF editor. Figure 11 shows on the right hand side the EEF property page for a selected FRM parameter. The EEF property pages a clearly more user friendly than the standard tabular eclipse property pages.

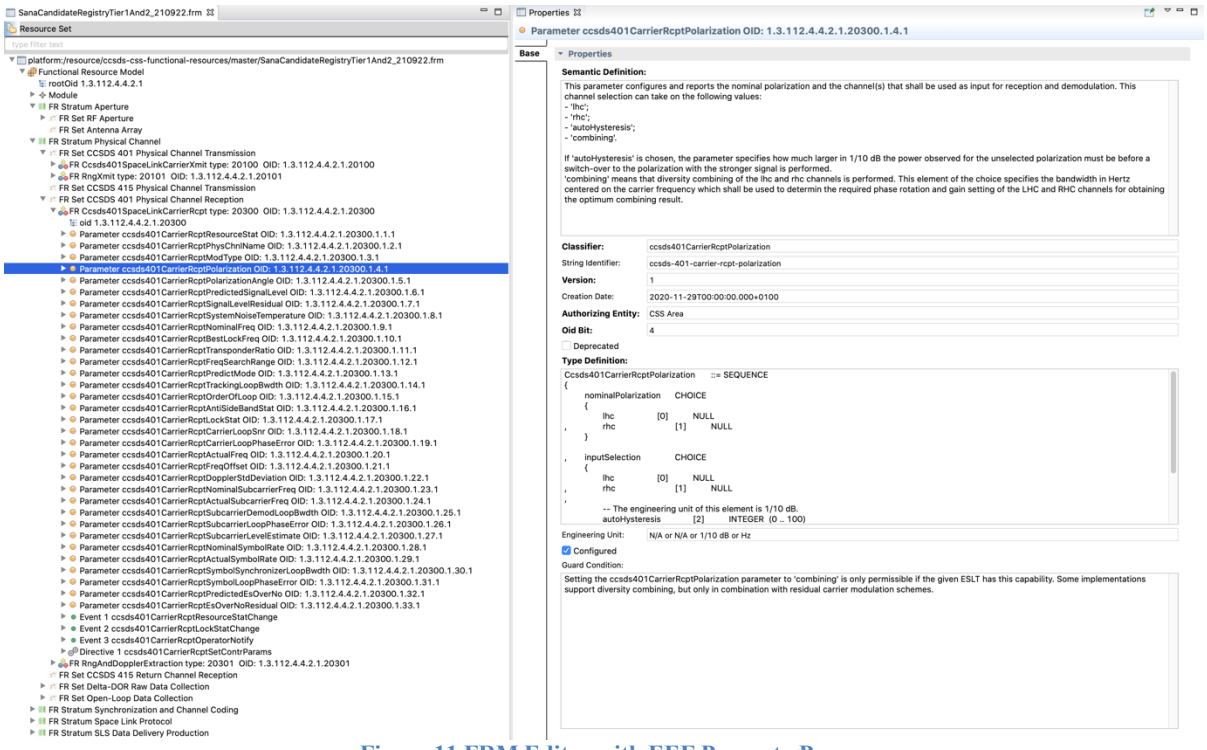


Figure 11 FRM Editor with EEF Property Page

5.3 FRM MET MODELS

The FRM editor is based on two meta models.

1. The Functional Resource Meta model itself (ecore)
2. The Type Definition Meta model (ecore)

For modularity (reuse) reasons two different ecore models have been used. Clearly the Functional Resource Meta Model imports the Type Definition Meta model and makes use of it. The structure of the two ecore models are shown below in Figure 12 and Figure 13.

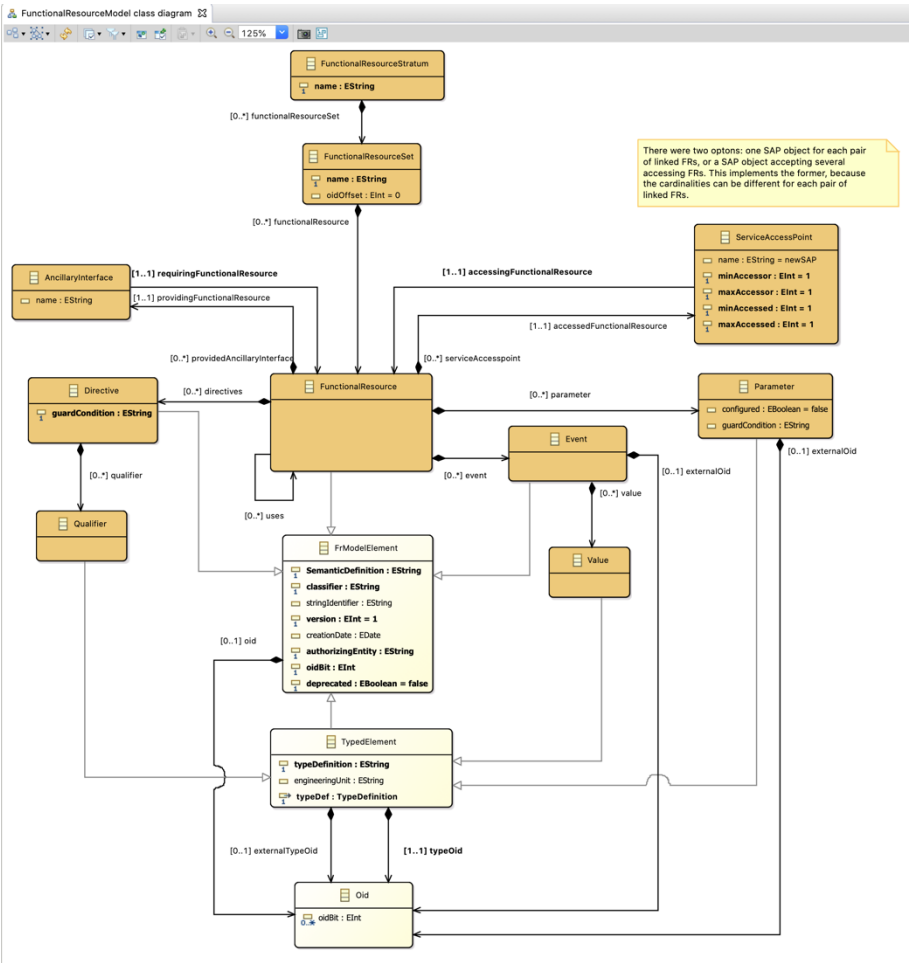


Figure 12 Functional Resource Meta Model (ecore)

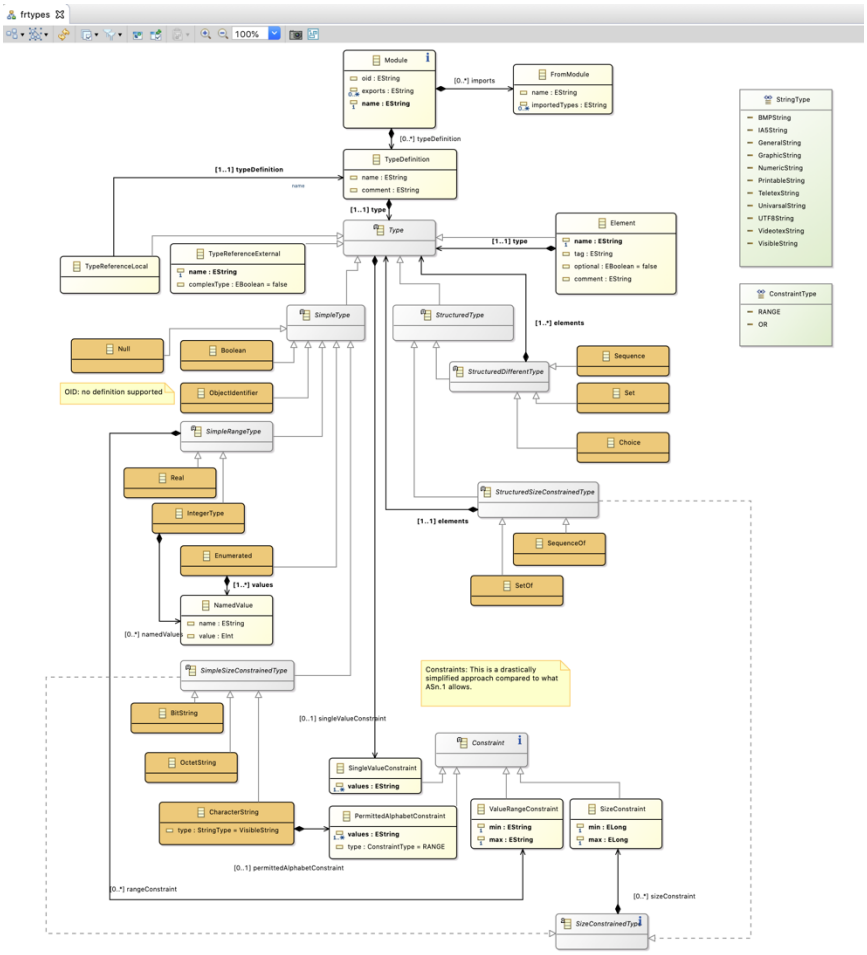


Figure 13 Type Definition Meta Model

5.4 FRM SPECIFIC TOOLS

All FRM specific functions like OID numbering have been implemented in a dedicated plug-in **ccsds.fr.model.tools.**

5.5 GRAPHICAL EDITING

For graphical editing eclipse Sirius has been used, see plug-in **ccsds.fr.model.tools.diagram.design**.

5.6 CHANGING THE FRM METAMODEL

If the FRM ecore model is updated:

- reload the genmodel and generate the model / edit / editor code
- initialize the EEF models from the genmodel (without deleting them)
- Run the tool 'Update EEF Model' to use a multiline edit control for semantic description (${workspace_loc:/ccsds.fr.model.edit/models/update_components.sh})

- use the eefgen to Generate the EEF architecture. It might be necessary to delete the generated code for elements removed from the ecore.
- Checkout the two OID related file generated by EEF to maintain the simple text widget for OID editing:
  `git checkout ccsds.fr.model.edit/src-gen/ccsds/FunctionalResourceModel/parts/forms/OidPropertiesEditionPartForm.java ccsds.fr.model.edit/src-gen/ccsds/FunctionalResourceModel/parts/impl/OidPropertiesEditionPartImpl.java`
- Global replace of *.heightHint = 80* with *.heightHint = 160* to have the semantic description in a text field of appropriate size.
- If a new top-level element has been included add two extension points for the top-level element in *ccsds.fr.model.edit/plugin.xml.*

5.7 SEPARATED MODELS FOR FRM AND FRM TYPES

The two ecore models for FRM and FRM types are separated for modularity reasons. The FRM ecore model imports the EMF Type model in the standard EMF way. Recipe to have EEF properties also working for FRM Type objects in the FRM editor:

- Generate the EEF architecture for FRM types as per EEF documentation. Actually the editor is not required, but may be useful.
- Duplicate the **`point="org.eclipse.ui.views.properties.tabbed.propertySections"`** in the *ccsds.fr.type.model.edit* plug-in manifest with **`contributorId="ccsds.FunctionalResourceModel.properties".`** Otherwise the property sections are not available in (i.e. contributed to) the FRM editor. The *contributorId* is actually the ID of the FRM editor.