POC EASE for Capella

Thales - MSBE

Exported on 04/06/2021

Table of Contents

In order to evaluate the implementation in the definition of Capella scripts with EASE, a POC has been realized.

Here are the results of this POC

1 Definition of the POC scope

In this POC, we will try to answer the following need:

> As a user, I want an Excel export with the Functional Chains, the functions involved and for each Function, the Property Values defined with PVMT, and the requirements attached with the Requirement add-on

2 Definition and architecture of the solution

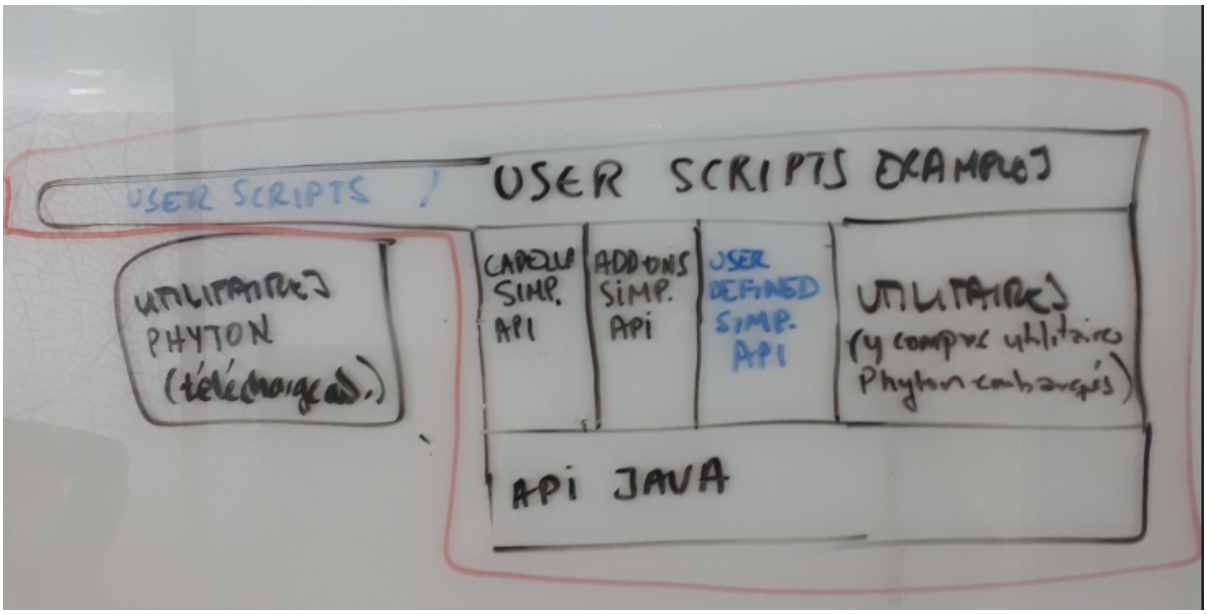
As a reminder, the solution consists in the use of EASE + PyDev + Python to define scripts for Capella.

In addition to that, the solution is based on a 3 layers architecture:

1. at the lowest level, Java APIs that allow to interoperate with Capella
2. in the middle, Python scripts defined by advanced users (quite technical)
3. at the highest level, Python scripts defined by lambda users (higher level)

In this diagram:

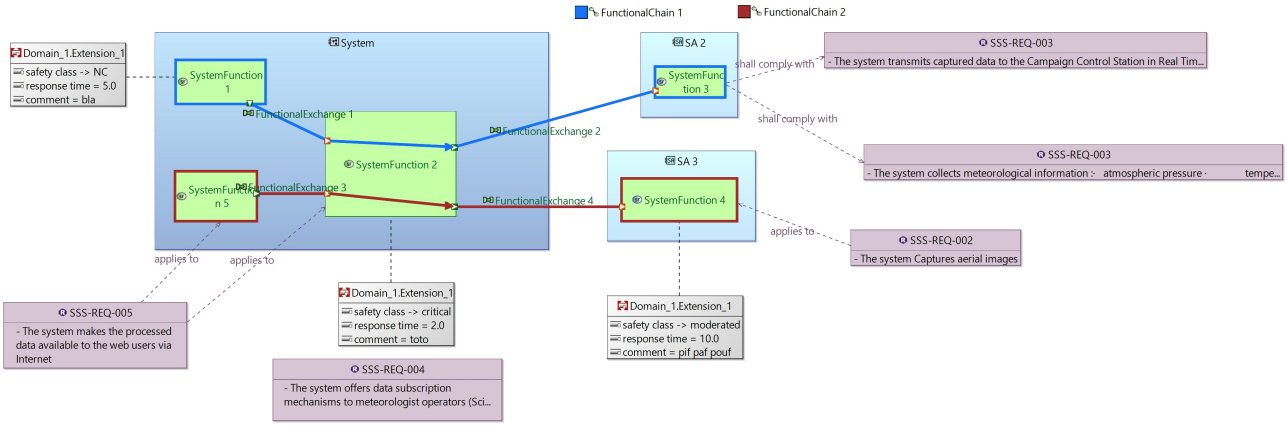
- Circled in red: the scope of the solution
- Blue: what is created by users



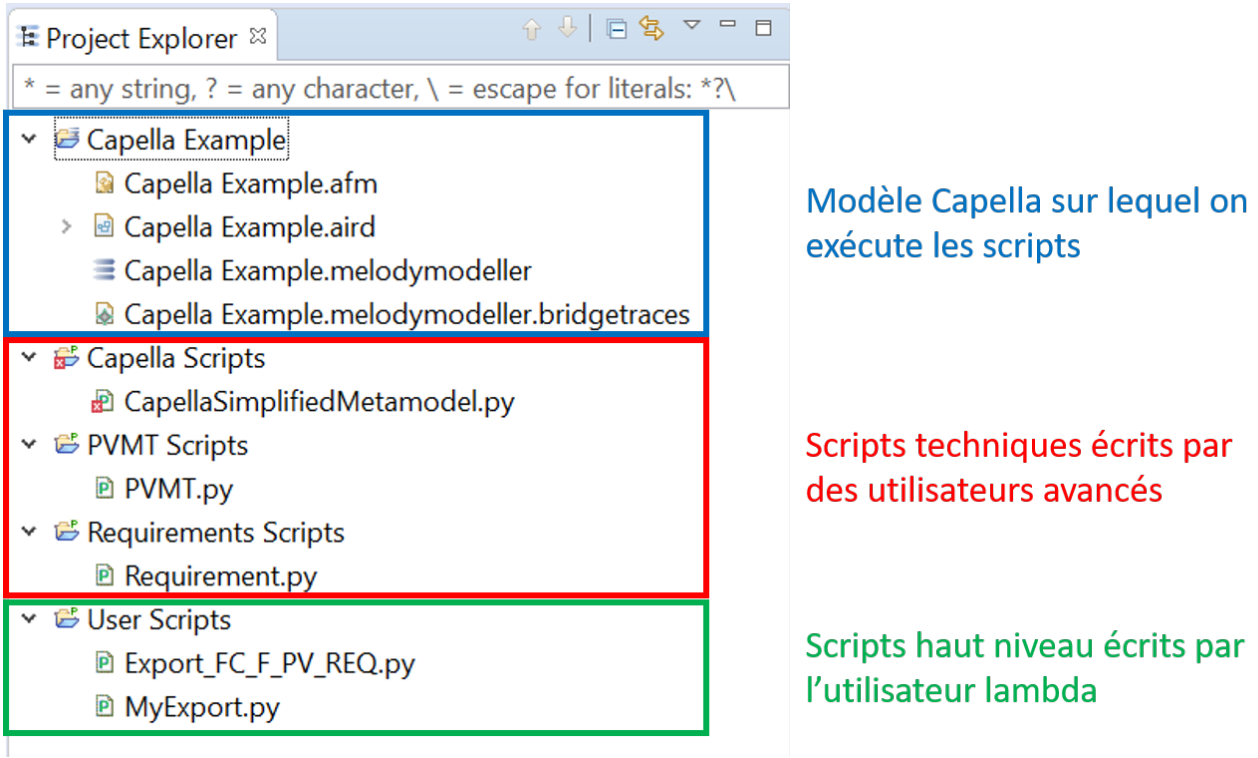
3 POC Implementation

3.1 Overview

The POC was implemented on this simple model:



Different scripts have been written to answer the need:



Concerning the technical scripts, it was decided to write 3 different scripts in different projects in order to separate what is linked to the core of Capella, and what is linked to the different add-ons (here PVMT and Requirements). The user will be able to use only the basic Capella script library, or use (or create for himself) a library linked to a specific add-on.

Concerning the user scripts, we have also separated them into 2 different scripts: one to define the information to be extracted from the model and the export in excel form, the other to execute this export on a specific Capella model with a path for the output excel file. Thus, if the user wants to perform this type of export on several different Capella models, he can call the export script several times with different configurations (input Capella model / output Excel file).

3.2 Java API

2 EASE modules have been defined for this POC: an EMF level module and a Capella level module.

The content of these modules was limited to the scope of the POC

EMF Module

```
1  package capella;
2
3  import org.eclipse.ease.modules.WrapToScript;
4  import org.eclipse.emf.common.util.BasicEList;
5  import org.eclipse.emf.common.util.EList;
6  import org.eclipse.emf.common.util.TreeIterator;
7  import org.eclipse.emf.ecore.EObject;
8
9  public class EMFModule {
10
11     public EMFModule() {
12     }
13
14     @WrapToScript
15     public Object getEAllContents(EObject obj) {
16         EList<EObject> result = new BasicEList<EObject>();
17         TreeIterator<EObject> it = obj.eAllContents();
18         while (it.hasNext()) {
19             EObject next = it.next();
20             result.add(next);
21         }
22         return result;
23     }
24
25 }
```

This first module is almost empty.

In fact, EASE allows to call directly Java methods when manipulating Java objects with scripts. However, PyDev does not offer any content assist on the Java methods that can be called. But in any case, PyDev does not offer any content assist on the EASE modules defined...

In our case, we decided not to redefine the methods that could be called directly on the Java objects.

The only method defined in this first module is getEAllContents which is also a Java method defined on EObjects (eAllContents()).

Nevertheless, for an unknown reason (typing problem with the Java Iterator ?) it was not possible to call directly the Java method eAllContents from the Python scripts...

Capella Module

```java
package capella;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Set;
import org.eclipse.core.runtime.NullProgressMonitor;
import org.eclipse.ease.modules.WrapToScript;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.sirius.business.api.session.Session;
import org.polarsys.capella.common.helpers.EObjectExt;
import org.polarsys.capella.core.data.capellamodeller.Project;
import org.polarsys.capella.core.data.capellamodeller.SystemEngineering;
import org.polarsys.capella.core.sirius.ui.helper.SessionHelper;
import org.polarsys.capella.common.ui.massactions.core.shared.helper.SemanticBrowserHelper;
import org.polarsys.capella.common.ui.toolkit.browser.category.ICategory;


import utils.DiagramSessionHelper;

public class CapellaModule {

    public CapellaModule() {
    }

    @WrapToScript
    public SystemEngineering getModelRoot(String fileURI) {
        fileURI = "platform:/resource/" + fileURI;
        URI uri = URI.createURI(fileURI);
        DiagramSessionHelper.setAirdUri(uri);
        Session session = DiagramSessionHelper.initSession();
        session.open(new NullProgressMonitor());
        Project rootSemanticElement = SessionHelper.getCapellaProject(session);
        return (SystemEngineering) rootSemanticElement.getOwnedModelRoots().get(0);
    }

    @WrapToScript
    public List<String> getAvailableSBQueries(EObject obj) {
        List<String> result = new ArrayList<String>();
        Collection<EObject> col = new ArrayList<EObject>();
        col.add(obj);
        Set<ICategory> SBQueries = SemanticBrowserHelper.getCommonObjectCategories(col);
        for (ICategory cat : SBQueries) {
            result.add(cat.getName());
        }
        return result;
    }

    @WrapToScript
    public List<EObject> getSBQuery(EObject obj, String query) {
        List<EObject> queryResult = new ArrayList<EObject>();
        Collection<EObject> col = new ArrayList<EObject>();
```

```
54          col.add(obj);
55          Set<ICategory> SBQueries = SemanticBrowserHelper.getCommonObjectCategories(col);
56          ICategory category = null;
57          for (ICategory cat : SBQueries) {
58              if (cat.getName().equals(query)) {
59                  category = cat;
60                  break;
61              }
62          }
63          if (category != null) {
64              for (Object object : category.compute(obj)) {
65                  if (object instanceof EObject) {
66                      queryResult.add((EObject) object);
67                  }
68              }
69          }
70          return queryResult;
71      }
72
73      @WrapToScript
74      public String getLabel(EObject obj) {
75          return EObjectExt.getText(obj);
76      }
77  }
```

This second module has a little more content:

- getModelRoot: allows to open a Capella model and to retrieve the SystemEngineering element from the path to the aird file present in the Capella workspace
- getAvailableSBQueries : allows to list the Semantic Browser queries available for an element according to its type (**not used in the POC**)
- getSBQuery : allows to get the result of a Semantic Browser query by providing a model element and the name of the query we are interested in
- getLabel : allows to get the Label (what is displayed in the Project Explorer) of a Capella element (**not used in the POC**)

3.3 Strategy for technical scripts

The idea of technical scripts is to hide the complexity of the technical metamodel to the end users.

To do this, Python supports object programming: you can define classes with methods that you can then call.

(the advantage is that by defining these classes, the user will be able to easily call the methods that are defined with the content assist)

This will allow to define a simplified metamodel which will be implemented by these Python classes.

(/!\ with the workload in the definition and the maintenance of this simplified metamodel...)

But Python is a weakly typed language. We don't always know the type of the variables we manipulate.

This is the case for example when you do a for loop on the elements of a list : Python (or at least PyDev) does not know how to define the type of the iteration variable.

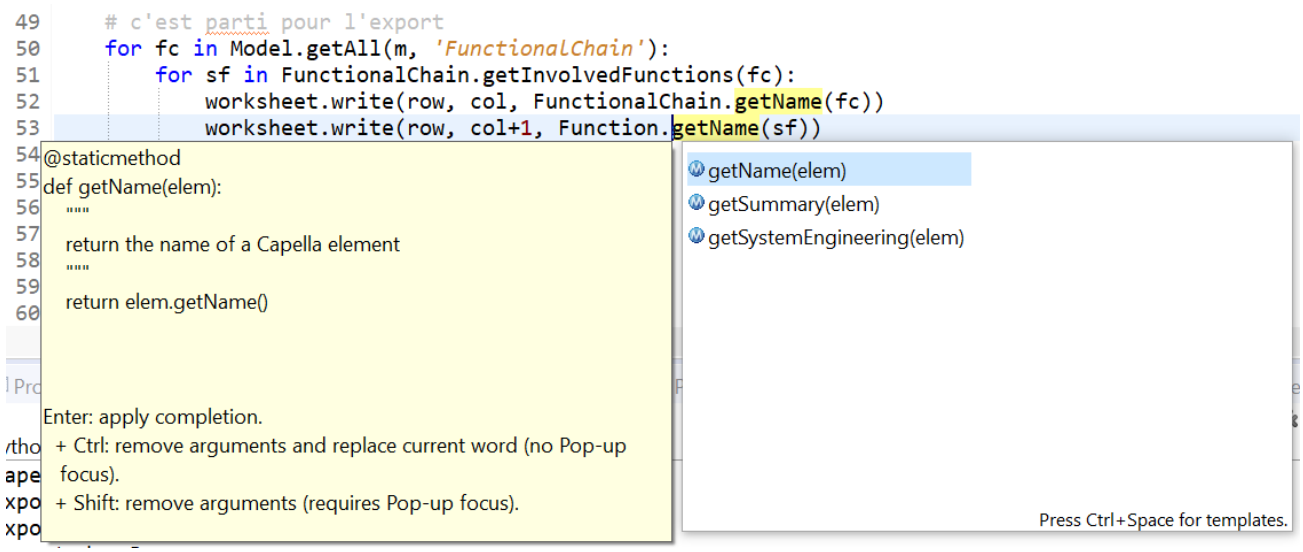
To overcome this problem, it was decided to implement Python classes with static methods.

So, if we have a function f and we want to retrieve its name, we can use the code :

```
Function.getName(f)
```

(we call the metaclass "Function" which then gives us access to its list of associated methods. We call getName by passing as a variable the function whose name we want to get)

Python and PyDev allow to describe the methods defined on the metaclasses in order to help the user to understand what he will do:



3.4 Capella Simplified Metamodel Script

```
'''
Created on 8 févr. 2021

@author: T0186002
'''

loadModule('/Capella/EMF')
loadModule('/Capella/Capella')

class CapellaElement():

    @staticmethod
    def getSystemEngineering(elem):
        """
        return the root System Engineering
        """
        if elem.eClass().getName() == 'SystemEngineering':
            return elem
        else:
            return CapellaElement.getSystemEngineering(elem.eContainer())

class NamedElement(CapellaElement):
    @staticmethod
    def getName(elem):
        """
        return the name of a Capella element
        """
        return elem.getName()

    @staticmethod
    def getSummary(elem):
        """
        return the Summary of a Capella element
        """
        return elem.getSummary()

class Model(NamedElement):
    @staticmethod
    def open(path):
        """
        return the System Enginnering element of a model by providing the path toward aird file
        for example se = Model.open('In-Flight Entertainment System/In-Flight Entertainment System.aird')
        """
        return getModelRoot(path)

    @staticmethod
    def getAll(model, type):
        """
        return all elements of the given type within the model
        """
```

```
        result = []
        # de manière étrange je n'arrive pas à utiliser "m.eAllContents()"
        # alors que je peux utiliser "m.eContents()"
        # un problème avec les iterator java peut être...
        allElements = getEAllContents(model)
        for elem in allElements :
            if (elem.eClass().getName() == type) :
                result.append(elem)
        return result

class FunctionalChain(NamedElement):
    @staticmethod
    def getInvolvedFunctions(fc):
        """
        return the list of involved functions
        """
        return fc.getInvolvedFunctions()

class Function(NamedElement):
    pass

class SystemFunction(Function):
    pass

class LogicalFunction(Function):
    pass

class PhysicalFunction(Function):
    pass
```

This script allows defining the Capella simplified metamodel. It was only partially defined in the frame of this POC.

It contains following definitions:

- CapellaElement
  - getSystemEngineering : get the parent SystemEngineering element of any model element
- NamedElement → CapellaElement
  - getName : get the name of an element
  - getSummary : get the summary of an element
- Model → NamedElement
  - open : get the SystemEngineering element from the path toward an aird file
  - getAll : get all elements of a given type
- FunctionalChain → NamedElement
  - getInvolvedFunctions : get the list of functions involved in a Functional Chain
- Function → NamedElement
- SystemFunction → Function
- LogicalFunction → Function
- PhysicalFunction → Function

3.5 PVMT Script

```
'''
Created on 17 févr. 2021

@author: T0186002
'''

include('workspace://Capella Scripts/CapellaSimplifiedMetamodel.py')
if False:
    from CapellaSimplifiedMetamodel import *

class PVMT ():

    @staticmethod
    def getPV(elem):
        """
        return the list of Property Values applied to an element
        """
        result = []
        for pvg in elem.getOwnedPropertyValueGroups():
            for pv in pvg.getOwnedPropertyValues():
                result.append(pv)
        return result

    @staticmethod
    def isPVDefined(elem, PVName):
        """
        return true if a Property with the name PVName is defined on the element elem
        """
        for pv in PVMT.getPV(elem):
            if pv.getName() == PVName:
                return True
        return False

    @staticmethod
    def getPVValue(elem, PVName):
        """
        return the value of the Property PVName applied to elem
        """
        for pv in PVMT.getPV(elem):
                if (NamedElement.getName(pv) == PVName):
                    if pv.eClass().getName() == 'EnumerationPropertyValue':
                        return pv.getValue().getName()
                    return pv.getValue()
```

This script allows to simplify the access to properties defined with PVMT.

It contains the following definitions:

- getPV : get the list of PVs defined on a model element
- isPVDefined : returns TRUE if a property with the defined name is already defined on the model element
- getPVValue : returns the value of a PV defined on a model element

3.6 Requirement Script

```
'''
Created on 17 févr. 2021

@author: T0186002
'''

loadModule('/Capella/Capella')

class Requirement():

    """
    /!\ ici on recupère les REQ en utilisant les requêtes du Semantic Browser.
    Hors l'add-on Requirement vient avec un bouton pour afficher ou non les REQ dans le Semantic Browser
    Dans l'implémentation actuelle, la fonction "getSBQuery" est sensible à ce bouton
    (si le bouton est désactivé, le script n'exporte pas les REQ)
    """

    @staticmethod
    def getOutgoingRequirement(elem):
        """
        from a model element, provide the list of requirements with an outgoing link
        """
        return getSBQuery(elem, 'Allocated Requirements')

    @staticmethod
    def getIncomingRequirement(elem):
        """
        from a model element, provide the list of requirements with an incoming link
        """
        return getSBQuery(elem, 'Allocating Requirements')

    @staticmethod
    def getRelationType(elem, req):
        """
        return the relation type linking a model element and a requirement
        """
        Relationfound = False
        for content in elem.eContents():
            if (content.eClass().getName() == 'CapellaOutgoingRelation'):
                if (content.getTarget() == req):
                    relation = content
                    Relationfound = True
                    break
        if Relationfound == False:
            for rel in req.getOwnedRelations():
                if (rel.getTarget() == elem):
                    relation = rel
                    Relationfound = True
                    break
        if (Relationfound):
```

```
51              return relation.getRelationType().getReqIFLongName()
52
53      @staticmethod
54      def getReqID(req):
55          """
56          return the ID of a requirement
57          """
58          for att in req.getOwnedAttributes():
59              if att.getDefinition().getReqIFLongName() == 'IE PUID':
60                  return att.getValue()
61
62      @staticmethod
63      def getReqText(req):
64          """
65          return the text of a requirement
66          """
67          return req.getReqIFText()
```

This script allows to simplify the access to the requirements defined with the Requirement add-on.

It contains the following definitions:

- getOutgoingRequirement : retrieve the requirements linked to a model element with an outgoing link
- getIncomingRequirement : allows to get the requirements linked to a model element with an incoming link
- getRelationType : retrieve the type of relationship between a model element and a requirement
- getReqID: retrieve the ID of a requirement
- getReqText: retrieve the text of a requirement

3.7 Export FC F PV REQ Script

```
'''
Created on 10 févr. 2021

@author: T0186002
'''

# les imports

import xlsxwriter

include('workspace://Capella Scripts/CapellaSimplifiedMetamodel.py')
if False:
    from CapellaSimplifiedMetamodel import *

include('workspace://PVMT Scripts/PVMT.py')
if False:
    from PVMT import *

include('workspace://Requirements Scripts/Requirement.py')
if False:
    from Requirement import *


def Export_FC_F_PV_REQ(modelPath, exportPath):

    # on initialise le fichier excel de sortie

    workbook = xlsxwriter.Workbook(exportPath)

    # on ouvre le modèle Capella

    m = Model.open(modelPath)
    print(Model.getName(m))

    # export des FC avec Fonctions

    print('export des FC')

    worksheet = workbook.add_worksheet('FC and Functions')

    row = 0
    col = 0

    # on écrit l'entête des colonnes
    worksheet.write(row, col, 'Functional Chain')
    worksheet.write(row, col+1, 'Function')
    row += 1

    # c'est parti pour l'export
    for fc in Model.getAll(m, 'FunctionalChain'):
        for sf in FunctionalChain.getInvolvedFunctions(fc):
```

```
            worksheet.write(row, col, FunctionalChain.getName(fc))
            worksheet.write(row, col+1, Function.getName(sf))
            row += 1

    # export des Fonctions avec PV

    print('export des PV')

    worksheet = workbook.add_worksheet('Functions and PV')

    # on construit la liste des PV

    allPV = []

    for sf in Model.getAll(m, 'SystemFunction'):
        for PV in PVMT.getPV(sf):
            PVName = NamedElement.getName(PV)
            if not (PVName in allPV):
                allPV.append(PVName)

    row = 0
    col = 0

    # on écrit l'entête des colonnes
    worksheet.write(row, col, 'Function')
    col += 1
    for PV in allPV:
        worksheet.write(row, col, PV)
        col += 1

    row += 1
    col = 0

    # on exporte les fonctions avec les PV
    for sf in Model.getAll(m, 'SystemFunction'):
        worksheet.write(row, col, SystemFunction.getName(sf))
        col += 1
        for pv in allPV:
            if PVMT.isPVDefined(sf, pv):
                worksheet.write(row, col, PVMT.getPVValue(sf, pv))
            col += 1
        row += 1
        col = 0

    # export des Fonctions avec REQ

    print('export des Req')

    worksheet = workbook.add_worksheet('Functions and Req')

    row = 0
    col = 0

    # on écrit l'entête des colonnes
    worksheet.write(row, col, 'Function')
    worksheet.write(row, col+1, 'Req Link Type')
```

```
108     worksheet.write(row, col+2, 'Requirement ID')
109     worksheet.write(row, col+3, 'Requirement Text')
110     row += 1
111 
112     # on exporte les fonctions avec les req
113     for sf in Model.getAll(m, 'SystemFunction'):
114         for req in Requirement.getOutgoingRequirement(sf) +
    Requirement.getIncomingRequirement(sf):
115             worksheet.write(row, col, SystemFunction.getName(sf))
116             worksheet.write(row, col+1, Requirement.getRelationType(sf, req))
117             worksheet.write(row, col+2, Requirement.getReqID(req))
118             worksheet.write(row, col+3, Requirement.getReqText(req))
119             row += 1
120 
121     workbook.close()
122 
```

This script defines the desired export. It generates an output excel file with 3 different tabs :

- the list of CF with the allocated functions
- the list of functions with the PV (cross matrix type)
- the list of functions with the requirements

The export to excel is done with the Python module XlsxWriter

PS: afterwards it turns out that XlsxWriter can only export to Excel. The module does not support opening and reading existing excel files. Other Python modules must exist for that...

3.8 MyExport Script

```
1  '''
2  Created on 17 févr. 2021
3 
4  @author: T0186002
5  '''
6 
7  include('Export_FC_F_PV_REQ.py')
8  if False:
9      from Export_FC_F_PV_REQ import *
10 
11 Export_FC_F_PV_REQ('Capella Example/Capella Example.aird', 'C:/Users/T0186002/Desktop/
   export.xlsx')
```

This script is the highest level.

It allows you to call the export script by entering the Capella model for which you want to export + the path and name for the output Excel file.

This is the script that is executed by the end user.

3.9 POC Result

Here is the excel file obtained from the scripts: export.xlsx[1]

1 https://wiki.corp.thales/download/attachments/1089995898/export.xlsx?api=v2&modificationDate=1613650764485&version=2

4 Conclusions

4.1 Synthesis

The POC allowed to implement the architecture principles of the desired solution.

The effort in the definition of the Java APIs should be quite limited because EASE natively allows to call the methods of the Java objects.

The definition of a simplified metamodel is possible using Python. It will be possible to create several script modules for the different extensions of Capella.

Data import to Capella has not been implemented in the POC but should not be a problem.

4.2 Difficulties of implementation and points of vigilance

Installation:

In the frame of this POC, installation of the environnement was not so easy.

Installation of an environnement with Capella + EASE + PyDev was not possible on a Thales computer (due to internet access restriction to Eclipse update sites).

Moreover, it was necessary to install Python on the computer which requires admin rights (was possible thanks to the Centre Logiciel).

Finally, installation of XlsxWriter module was not easy neither. The main installation mode for Python modules is online (pip command which directly get the module online) but is blocked once again by internet access rights. Installation with a tar.gz (tarball) archive also required a special configuration (--user) to avoid installation in system files (which requires asmin rights).

It is mandatory to provide a simple installation to users (in a few steps) which does not requires admin rights, nor internet access. (possibility to have a portable version of Python and its modules ?)

Documentation:

- *XlsxWriter Installation :* Getting Started with XlsxWriter — XlsxWriter Documentation[2]
- *Specific installation to avoid admin rights issue :* Installing Packages — Python Packaging User Guide[3]

Referencing between scripts:

Scripts referencing is not easy neither.

First, due to the fact that we use EASE and not directly Python, Python imports does not natively works.

We have to cheat with such command lines so that import wotks with EASE and PyDev content assist

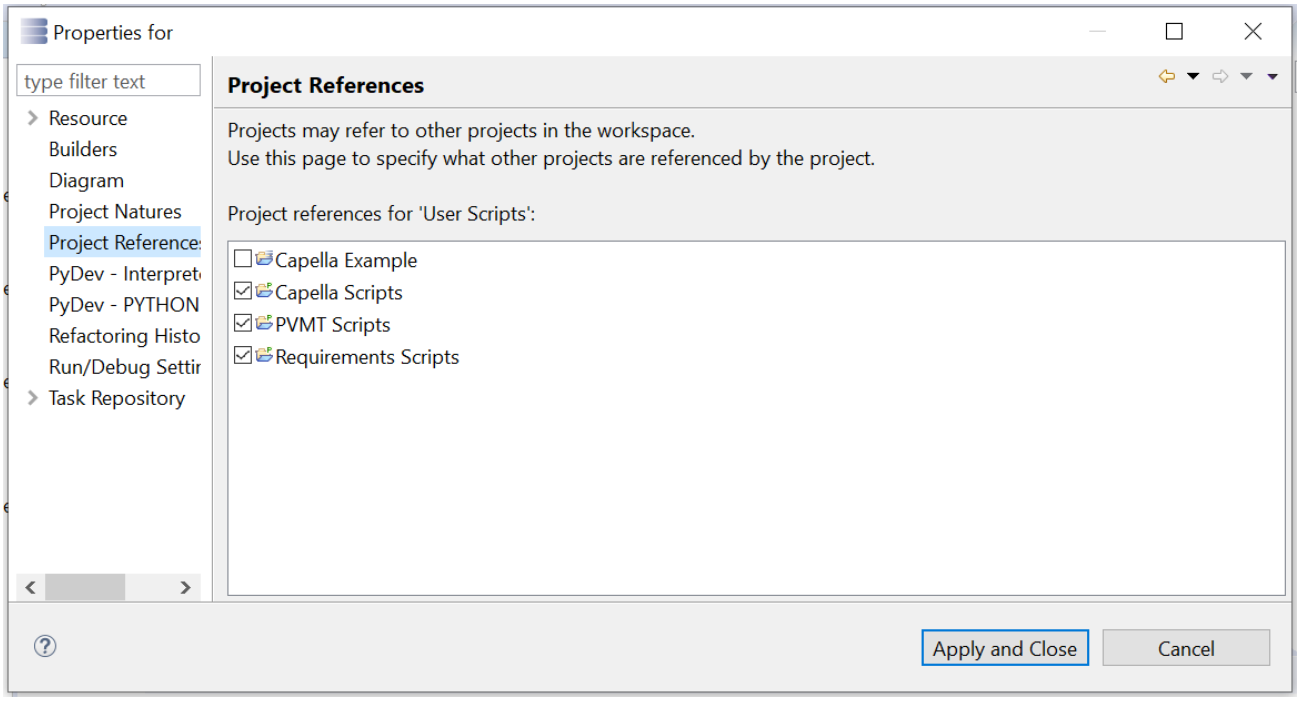
PS : PyDev identifies this line as an error…

2 https://xlsxwriter.readthedocs.io/getting_started.html#installing-xlsxwriter
3 https://packaging.python.org/tutorials/installing-packages/#installing-to-the-user-site

```
include('workspace://Capella Scripts/CapellaSimplifiedMetamodel.py')
if False:
    from CapellaSimplifiedMetamodel import *
```

Finally, as Python scripts have been defined in different Projets, it is required to reference other projects.

For this, we need to switch to the PyDev perspective and find the project properties:



This change of perspective + the fact that we need to reference eclipse projects makes this operation not easy for end-users...

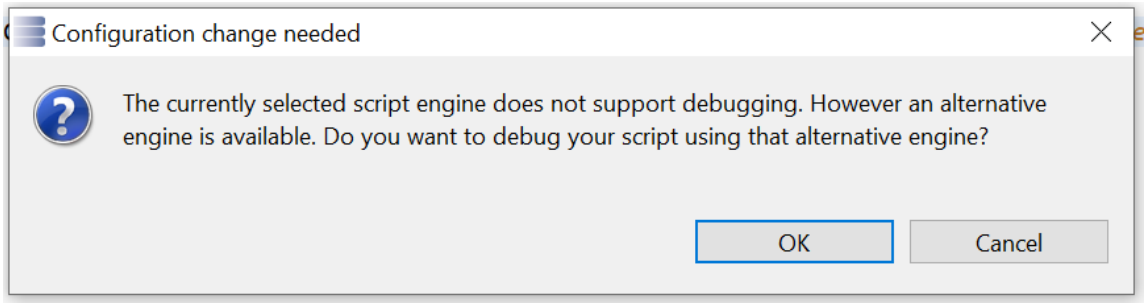
Usage:

Technical scripts definition requires a fine knowledge of Capella metamodel as the user will not benefit from any content assist.

The accessibility from system engineers entirely relies (except from Python knowledge) from the definition of a simplified metamodel and the maintenance of this simplified metamodel.

Debug Mode:

I was not able to use debug mode during the POC because of an error:



We need to secure this point as it will be important to debug complex scripts

5 Conformity with Requirements

Requirement	Solution
REQ1: a System Engineer shall be able to define scripts to pull and push data into the model • Here the System Engineering skills include: knowledge of the Arcadia concepts and relations, but no knowledge of the Capella metamodel, and some algorthmics skills in very well known high-level languages (such as Matlab, VBA or Phyton), but not Java.	**OK if level 2 scripts provide the right level of information**
REQ2: a System Engineer shall be able to easily share, deploy and use the scripts • Easy means: avoid steps of compilation, packaging and deployment	**OK that's the purpose of scripts**
REQ3: a System Engineer shall be able to write scripts in a common language (not specific to Capella) • For this purpose, Python is a good candidate but not the only option	**OK if we use Python**
REQ4: a System Engineer shall be able to use scripts that hide complexity of the technical metamodel of Capella • Similar to what he sees in the Semantic Browser. However it should be noted that Semantic Browser is not complete and may miss some important relation.	**Should be OK if we build the full library of technical Python scripts which defines a simplified datamodel for Capella**
REQ5: an Advanced System Engineer shall be able to access any important information / relation in the Capella model, and create scripts libraries to be used by Systems Engineers • Here an Advanced System Engineer has a deeper knowledge of the Capella metamodel and high-level languages programming skills	**OK - Java API shall be generic enough + Ease allow to call any method defined on Java objects**
REQ6: The solution shall support extensibility of Capella • Extensibility means that any extension of the Capella metamodel (including those through viewpoints / add-ons) shall be supported natively	**OK - this is performed by having a set of technical scripts for add-ons which enrich Capella metamodel**

Requirement	Solution
REQ7: The solution shall be maintainable by TCE in the long term with minimum effort	**Should be OK (depends on the different levels of scripts / API)**
REQ8: The solution shall not require migration between 2 Capella versions if the metamodel is not modified	**OK if Java API are generic enough**
REQ9: The solution shall support both Capella and Team for Capella	**Should be OK**
REQ10: The solution shall support execution in the Capella environment and execution in batch mode (headless)	**OK - ease normally support batch mode**
REQ11: The solution shall support various export formats (including but not limited to csv)	**OK - just use Python libraries**
REQ12: The solution shall support previous versions of Capella • Starting from 1.2.x ?	**To be validated regarding compatibility with Capella 1.2.x**
REQ13: a System Engineer shall be able to install the full solution easily, within a few steps and without requiring internet access nor administrator rights	**To be validated** **I had to install Python by myself on my computer (may require admin rights)** **+ How to install Python modules?**
REQ14: The solution shall provide the capability to display result of exports in Capella and support navigability with Capella views (Project Explorer, Semantic Browser, Diagrams...) • be able to display elements in search view for example	**Should be possible providing the right Java API**