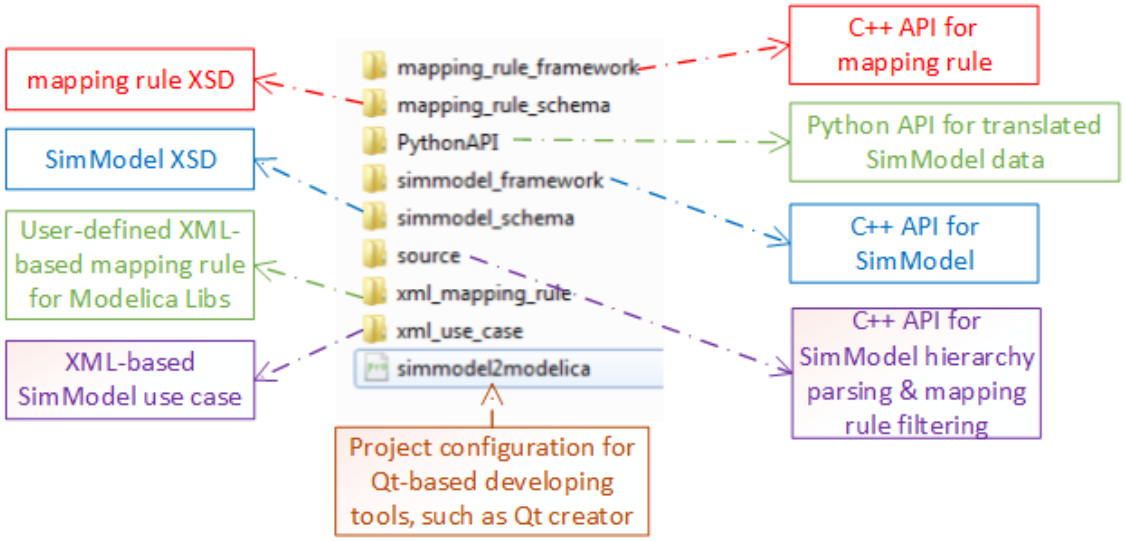
SimModel2Modelica Transformation API for Python

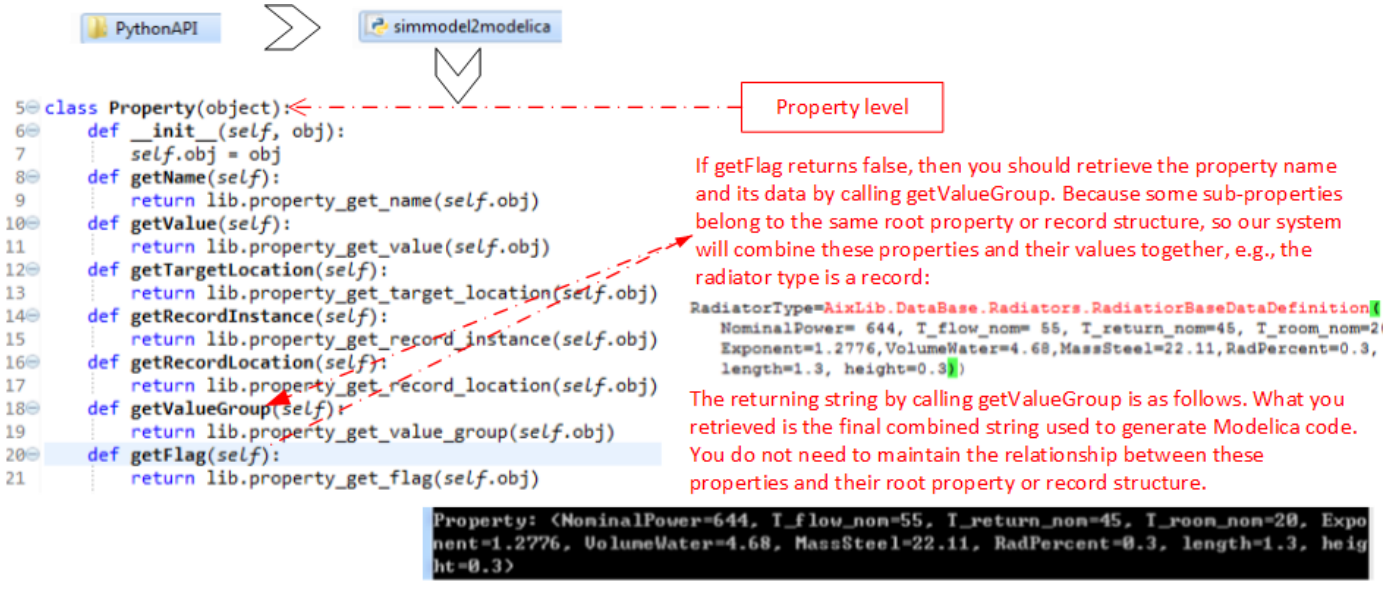
Jun Cao[1], Reinhard Wimmer[1], Tobias Maile[1], James O'Donnell[2], Christoph van Treeck[1]
[1]Institute of Energy Efficient Building E3D, RWTH Aachen University, Germany
[2]School of Mechanical and Materials Engineering and Electricity Research Centre, University College Dublin, Ireland

In this quick guideline, we provide you the instructions compiling our SimModel to Modelica transformation system and the illustration concerning the usage of the Python API of our system for generating your code in Modelica side.

1) The code structure of our project is:



2) In this section, we will briefly show you the Python API for retrieving Modelica model data translated from SimModel side based on the user-defined library-specific data mapping rules.




```
class Property(object):
    def __init__(self, obj):
        self.obj = obj
    def getName(self):
        return lib.property_get_name(self.obj)
    def getValue(self):
        return lib.property_get_value(self.obj)
    def getTargetLocation(self):
        return lib.property_get_target_location(self.obj)
    def getRecordInstance(self):
        return lib.property_get_record_instance(self.obj)
    def getRecordLocation(self):
        return lib.property_get_record_location(self.obj)
    def getValueGroup(self):
        return lib.property_get_value_group(self.obj)
    def getFlag(self):
        return lib.property_get_flag(self.obj)
```

```
class Component(object):
    def __init__(self, obj):
        self.obj = obj
    def getTargetName(self):
        return lib.component_get_target_name(self.obj)
    def getTargetLocation(self):
        return lib.component_get_target_location(self.obj)
    def getProperty(self, id):
        return lib.component_get_property(self.obj, id)
    def getPropertyNumber(self):
        return lib.component_get_property_number(self.obj)

class RuleData(object):
    def __init__(self):
        self.obj = lib.rule_data_init()
    def getComponent(self, id):
        return lib.rule_data_get_component(self.obj, id)
    def getComponentNumber(self):
        return lib.rule_data_get_component_number(self.obj)
```

Component level

A list of Components

Run the Python API wrapper: in the command line or shell terminal, call Python executes the simmodel2modelica.py file located in the folder 'PythonAPI', like:

```
C:\>cd C:\Research\EnEff-BIM\source_code\simmodel2modelica\PythonAPI

C:\Research\EnEff-BIM\source_code\simmodel2modelica\PythonAPI>Python simmodel2mo
delica.py_
```

Then the translated data of components and their internal properties will be output as follows:

```
Component 0: AixLib.HVAC.Sources.Boundary_p boundary_p0
Property: p=1e5
Component 1: AixLib.HVAC.HeatGeneration.Boiler boiler1
Property: Q_flow_max=1300
Property: Volume=0.00999999977648
Property: volume(T(start=328.15, fixed=true))
Component 2: Modelica.Blocks.Sources.Constant const2
Component 3: AixLib.HVAC.Radiators.Radiator radiator3
Property: volume(V=0.00467999977991, T(start=303.1499938965, fixed=true))
Property: (NominalPower=644, T_flow_nom=55, T_return_nom=45, T_room_nom=20, Expo
nent=1.2776, VolumeWater=4.68, MassSteel=22.11, RadPercent=0.3, length=1.3, heig
ht=0.3)
Component 4: AixLib.HVAC.Pumps.Pump pump4
Property: V_flow_max=-2.14748e+09
Property: start=0.0000112961579362
Property: start=1
Component 5: Modelica.Blocks.Sources.BooleanPulse NightSignal5
Property: period=86400
```

Component level

Property level

Attention: this Python API wrapper needs 64-bit Python environment to load the pre-compiled SimModel2Model dll libraries. Besides this, you do not need to install any other external tools or libraries to run it.

3) This section we show you the instructions for compiling the source code. You can follow them step-by-step in order to generate the .dll library for Python API.

- Step 1: You need to install the XML binding parser CodeSynthesis XSD 4.0 on the target operating system (OS), e.g., Windows 7. The XML binding parser will convert SimModel XSD schema into C++ APIs. Our system needs it to re-generate the C++ APIs when the structure of SimModel XSD changed or load the SimModel data from the use case. You can download it from here: http://www.codesynthesis.com/products/xsd/download.xhtml
  The old version of CodeSynthesis XSD such as version 3.3 will cause some inbuilt bugs for the SimModel API when linking it to Qt platform. We can fix these bugs manually before switching into version 4.0. However, this way is not recommended for non-experienced programmer in C++ and Qt. For the installation of CodeSynthesis XSD 4.0, e.g., on Windows platform, you can download the binary program xsd-4.0.msi then install the package. There are two issues we need to be careful: a) the directory folder you select to install CodeSynthesis should not contain any space characters in its name. For example, the folder 'C:\Program Files (x86)' is not a good location for us to install the package. Because there are two white spaces inside the folder name 'Program/ Files/ (x86)'. Normally this special character will work fine for the CodeSynthesis XSD on the Windows platform. However, this time we need to integrate CodeSynthesis XSD into the Qt compiler, mingw, the Windows version of g++. And until yet, CodeSynthesis does not publish the binary package for supporting mingw. It means we need to compile the source code of its internal XML parser Xerces on Windows by calling mingw based on Linux environment simulation tools, such as Cygwin or Msys. b) when you configure the installation packages of CodeSynthesis XSD during the installation process, unselect the header files of Xerces for Visual Studio.
- Step 2: after installing CodeSynthesis XSD, we need to install Qt library and its developing toolkits. For our project, you need to install the 64-bit Qt packages. You can download it via this link: http://sourceforge.net/projects/qtx64/ , then install the whole developing packages, i.e., Qt libraries and x86_64-w64-mingw32 into your system. The same point for you: do not select the directory with special characters, i.e., white space, as the target installation folder.
- Step 3: if everything works fine, then we need to install Cygwin or Msys on your OS for compiling the binary code of xerces-c-3.1.1 library located in the CodeSynthesis XSD installation folder. The installation of Cygwin or Msys is not too hard, but out of the scope of this short guideline. We will add detail info later. What you need to install on these Linux simulation environment is some g++ based developing toolkits, such as autoconf, make, g++ compiler, etc.
- Step 4: open the terminal shell of Cygwin or Msys, then change your working directory into the target installation folder of CodeSynthesis XSD, e.g., 'C:\xsd\xsd4.0'. You will see a subfolder named 'xerces-c-3.1.1', which is the source code of CodeSynthesis XSD internal XML file parser.
  a) Before compiling the source code, execute the following directive in the terminal shell in order to set the path of your mingw compiler:
  export PATH=C:/Qt/qt-5.4.0-x64-mingw492r0-sjlj/mingw64/bin$PATH
  Here C:/Qt/qt-5.4.0-x64-mingw492r0-sjlj is the installation directory of my mingw compiler
  b) Then change your directory into the subfolder xerces-c-3.1.1:
  cd xerces-c-3.1.1 (i.e., C:\xsd\xsd4.0\xerces-c-3.1.1)
  and execute the following command:

./configure --enable-transcoder-windows --enable-netaccessor-winsock --disable-shared CXXFLAGS=-O2 CFLAGS=-O2

c) Go to the subfolder src:

cd src (i.e., C:\xsd\xsd4.0\xerces-c-3.1.1\src)

type command:

make

- Step 5: If no errors output in Step 4, then we can continue to install the Qt developing tools. Here we choose Qt creator. You can download it via the link: http://www.qt.io/download-open-source/

  While you install the package, only necessary to install Qt creator, because we do not need the other 32-bit Qt libraries attached in this community package.
- Step 6: locate to the root folder of our project 'simmodel2modelica', use a text editor to open the project configuration file 'simmodel2modelica.pro', and reset the directory of following marcos:

  ```
  INCLUDEPATH +=  "C:\xsd\xsd4.0\xerces-c-3.1.1\src" \
                  "C:\xsd\xsd4.0\include"
  ```

  Here the 1st line 'C:\xsd\xsd4.0\xerces-c-3.1.1\src' is the location of your Xerces library. It is located inside the CodeSynthesis XSD folder.

  The 2nd line 'C:\xsd\xsd4.0\include' is the head file of your CodeSynthesis XSD. Also located in the CodeSynthesis XSD installation folder. Reset these values by your target directory.

  In addition, reset the directory for the static library generated after we compiling the source code in Step 4.

  ```
  LIBS += -L"C:\xsd\xsd4.0\xerces-c-3.1.1\src\.libs" -lxerces-c
  ```
- Step 7: now you can left click on file 'simmodel2modelica.pro', the Qt creator will be loaded to open the project. Follow the info on Qt creator, configure the directory of your 64-bit mingw compiler and Qt libs we installed in Step 2. The latest Qt creator should find these tools by itself.
- Step 8: In the Qt creator, go to menu 'Build', then build the project. When the building is finished, you can find the simmodel2modelica.dll in the Debug or Release folder. Copy it into the subfolder PythonAPI, then you can use it as we showed in section 2.

  Attention:

  1) if you find above procedure is too complicated, then just use the pre-compiled dll library we provide in the PythonAPI folder.

  2) There is a bug in current Qt creator: if you load/include a very large head file or library in Qt creator, sometimes the Qt creator will crash due to the failure memory allocation for the function/method parsing, reminding and auto-completion. So if you wanna compose the source code of our project, please use the other code editor, such as Editplus or Visual Studio to modify the source code. And Qt creator right now will only be used to compile the project.