TANGO

Connecting things together

Pogo

The TANGO device class code generator

EUROPEAN SYNCHROTRON RADIATION FACILITY

Pascal Verdier

http://www.tango-controls.org/

Pogo TANGO code generator

- **A programmer point of view. Not a user point of view**
- **Brief history:**
  - Pogo is a java TANGO device class generator. It is able to generate C++, Java and Python code.
  - It has been started to be written in 2000 using its own code parsing until Pogo-6.2.5a
  - From Pogo-7.0.0, it has been re written completely with Eclipse plugin *OpenArchitectureWare* (OAW) based on ***Xtext*** to defined project and ***Xpand*** to generate code.
  - Until Pogo-8.1.0, it uses ***Xtend*** to generate code.
  - Until Pogo-9.5.0, it has been ported to Maven and IntelliJ Idea (Thanks to Igor).
  - Last distributed release is 9.6.31 (Java 8)
  - Last release 9.7.2 (Java 11 Thanks to Damien)

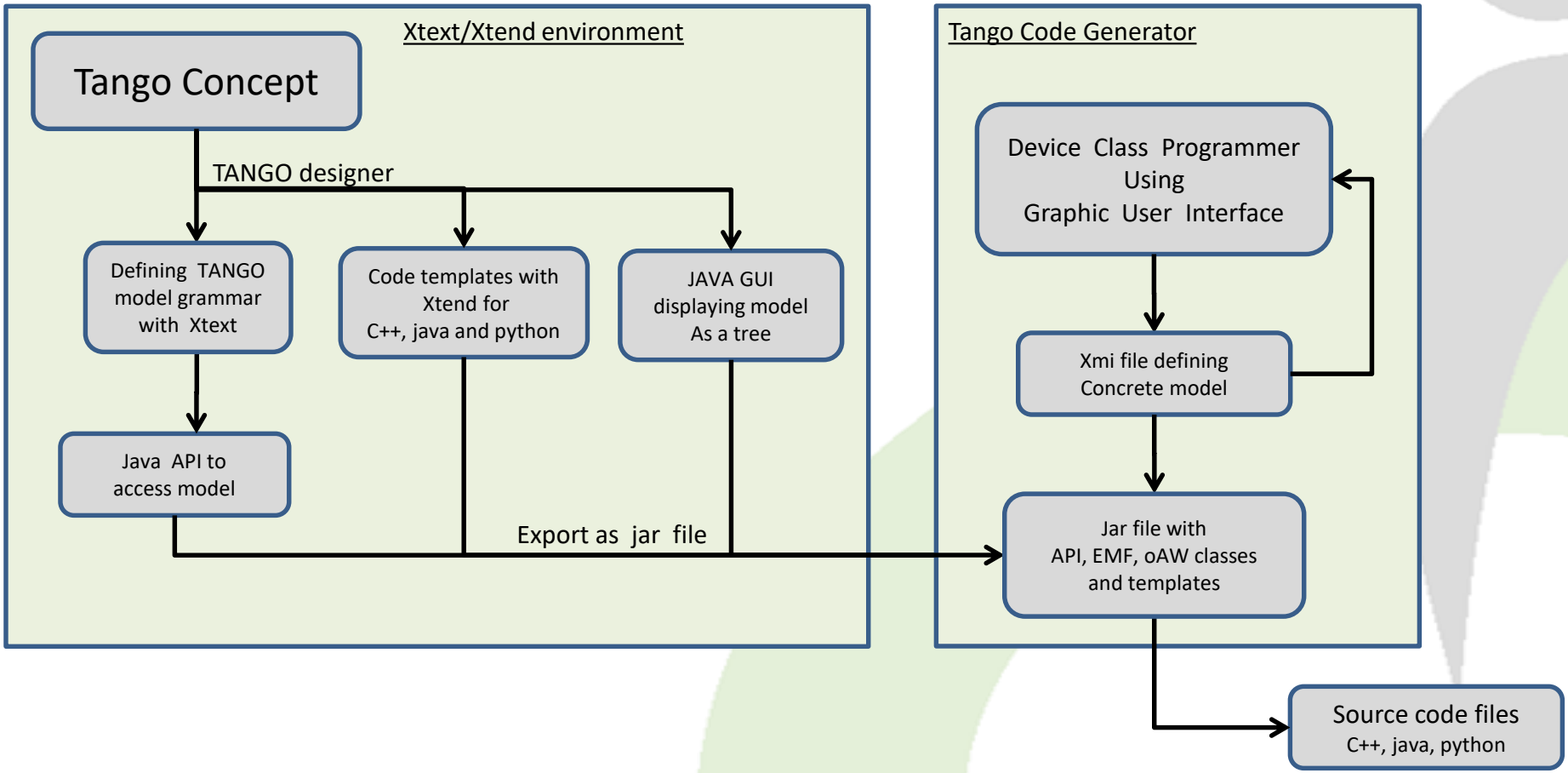
- **Wikipedia:**
  - ***Xtext*** is an open-source software framework for developing programming languages and domain-specific languages (DSL: computer language specialized to a particular application domain).
  - ***Xtend*** is a general-purpose high-level programming language for the Java Virtual Machine.

- **Remark:**

I did not manage Python code. It has been managed by Sebastien from Nexeya.

Pogo TANGO code generator

Workflow Principle:



Pogo TANGO code generator

- **Pogo project directories:**

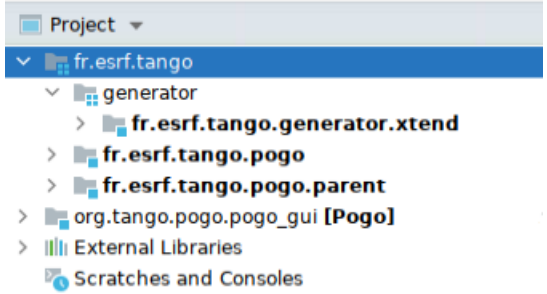
Directory	Usage
docs/	Old Pogo HTML documentation
fr.esrf.tango.generator.xtend/	Code to be generated definition (Xtend)
fr.esrf.tango.pogo/	Device class model (Xtext)
org.tango.pogo.pogo_gui/	Pogo GUI
fr.esrf.tango.pogo.parent/	pom.xml file to generate jar using maven
lib/	Protected regions management
fr.esrf.tango.pogo.ui/	Used by workflow
fr.esrf.tango.pogo.build/	Used by workflow

- **To start:**

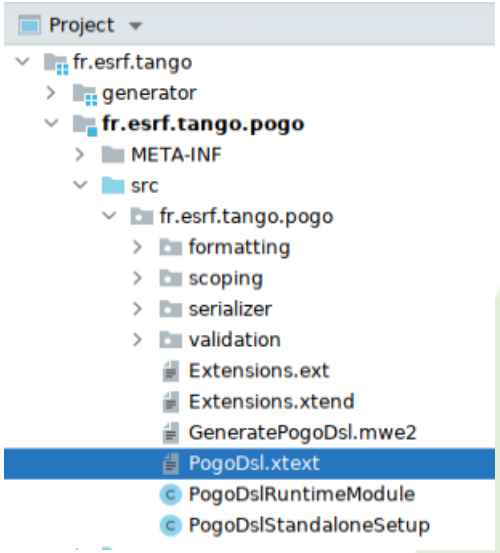
  See wiki written by Igor: https://github.com/tango-controls/pogo/wiki

Pogo TANGO code generator

- **Pogo project packages:**



- **Defining the class model**



Pogo TANGO code generator

```
PogoDeviceClass:
   "deviceclass" name=ID (isAbstract?="abstract")? ("extends" baseClass=[PogoDeviceClass])?
  "{"
                pogoRevision = ID                                       // To check Pogo revision when xmi has been created
                institute = ID                                          // For future specific features
                "description:"                                          // Class information
                        (description=ClassDescription)

                "classProperties:"                                      // Class property list
                        (classProperties+=Property)*
                "deviceProperties:"                                     // Device Property list
                        (deviceProperties+=Property)*

                "commands:"                                             // Device static command list
                        (commands+=Command)*
                "dynamicCommands:"                                      // Device dynamic command list
                        (dynamicCommands+=Command)*
                "attributes:"                                           // Device static attribute list
                        (attributes+=Attribute)*
                "dynamicAttributes:"                                    // Device dynamic attribute list
                        (dynamicAttributes+=Attribute)*
                "forwardedAttributes:"                                  // Device forwarded attribute list
                        (forwardedAttributes+=ForwardedAttribute)*
                "pipes:"                                                // Device pipe list
                        (pipes+=Pipe)*
                "states:"                                               // Device state list
                        (states+=State)*

                preferences = Preferences                               // Preferences (for programmer, for site or at run time)
                "additionalFiles:"                                      // Programmer's additional files to be added in Makefile (utilities, threads,...)
                        (additionalFiles+=AdditionalFile)*
                "overlodedPollPeriodObject:"                            // Object list (Command or Attribute) where polling period has been overloaded
                        (overlodedPollPeriodObject += OverlodedPollPeriodObject)*
        "}";
```

Pogo TANGO code generator

```
PogoDeviceClass:
   "deviceclass" name=ID (isAbstract?="abstract")? ("extends" baseClass=[PogoDeviceClass])?
  "{"
                pogoRevision = ID                                          // To check Pogo revision when xmi has been created
                institute = ID                                             // For future specific features
                "description:"                                             // Class information
                              (description=ClassDescription)

                "classProperties:"                                         // Class property list
                              (classProperties+=Property)*
                "deviceProperties:"                                        // Device Property list
                              (deviceProperties+=Property)*

                "commands:"                                                // Device static command list
                              (commands+=Command)*
                "dynamicCommands:"                                         // Device dynamic command list
                              (dynamicCommands+=Command)*
                "attributes:"                                              // Device static attribute list
                              (attributes+=Attribute)*
                "dynamicAttributes:"                                       // Device dynamic attribute list
                              (dynamicAttributes+=Attribute)*
                "forwardedAttributes:"                                     // Device forwarded attribute list
                              (forwardedAttributes+=ForwardedAttribute)*
                "pipes:"                                                   // Device pipe list
                              (pipes+=Pipe)*
                "states:"                                                  // Device state list
                              (states+=State)*

                preferences = Preferences                                  // Preferences (for programmer, for site or at run time)
                "additionalFiles:"                                         // Programmer's additional files to be added in Makefile (utilities, threads,...)
                              (additionalFiles+=AdditionalFile)*
                "overlodedPollPeriodObject:"                               // Object list (Command or Attribute) where polling period has been overloaded
                              (overlodedPollPeriodObject += OverlodedPollPeriodObject)*
      "}";
```

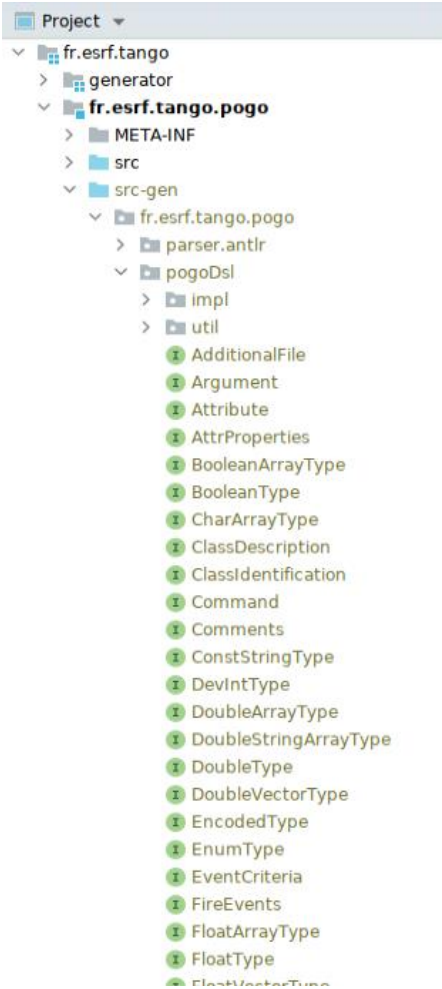
Pogo TANGO code generator

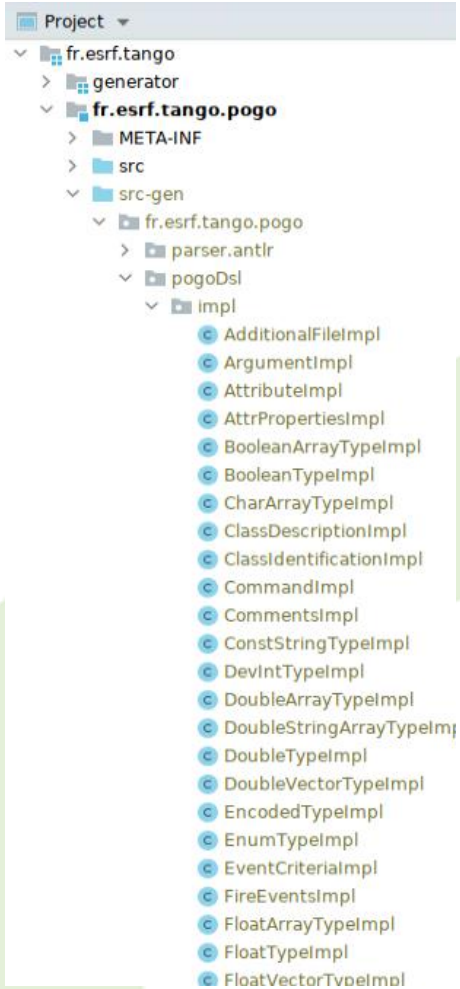
```
//
//          Command Definition
//
Command:
            name        = ID
            argin       = Argument
            argout      = Argument
            description = STRING
            status      = InheritanceStatus
            execMethod  = STRING
            displayLevel = DisplayLevel
            polledPeriod = STRING
            isDynamic   = Boolean
            "excludedStates:"
                        (excludedStates+=STRING)*
;

Argument:   type=Type description=STRING;

DisplayLevel:   "OPERATOR" | "EXPERT";

InheritanceStatus:
            abstract     = Boolean      // is abstract or inherited from abstract
            inherited    = Boolean      // is inherited
            concrete     = Boolean      // is concrete or overloaded
            concreteHere = Boolean      // is concrete or overloaded in this class
            hasChanged   = STRING;      // What has changed (dataType, attType, RWtype)
```

Pogo TANGO code generator

Compiling fr.esrf.tango.pogo/src/PogoDsl.xtext generates 2 packages:

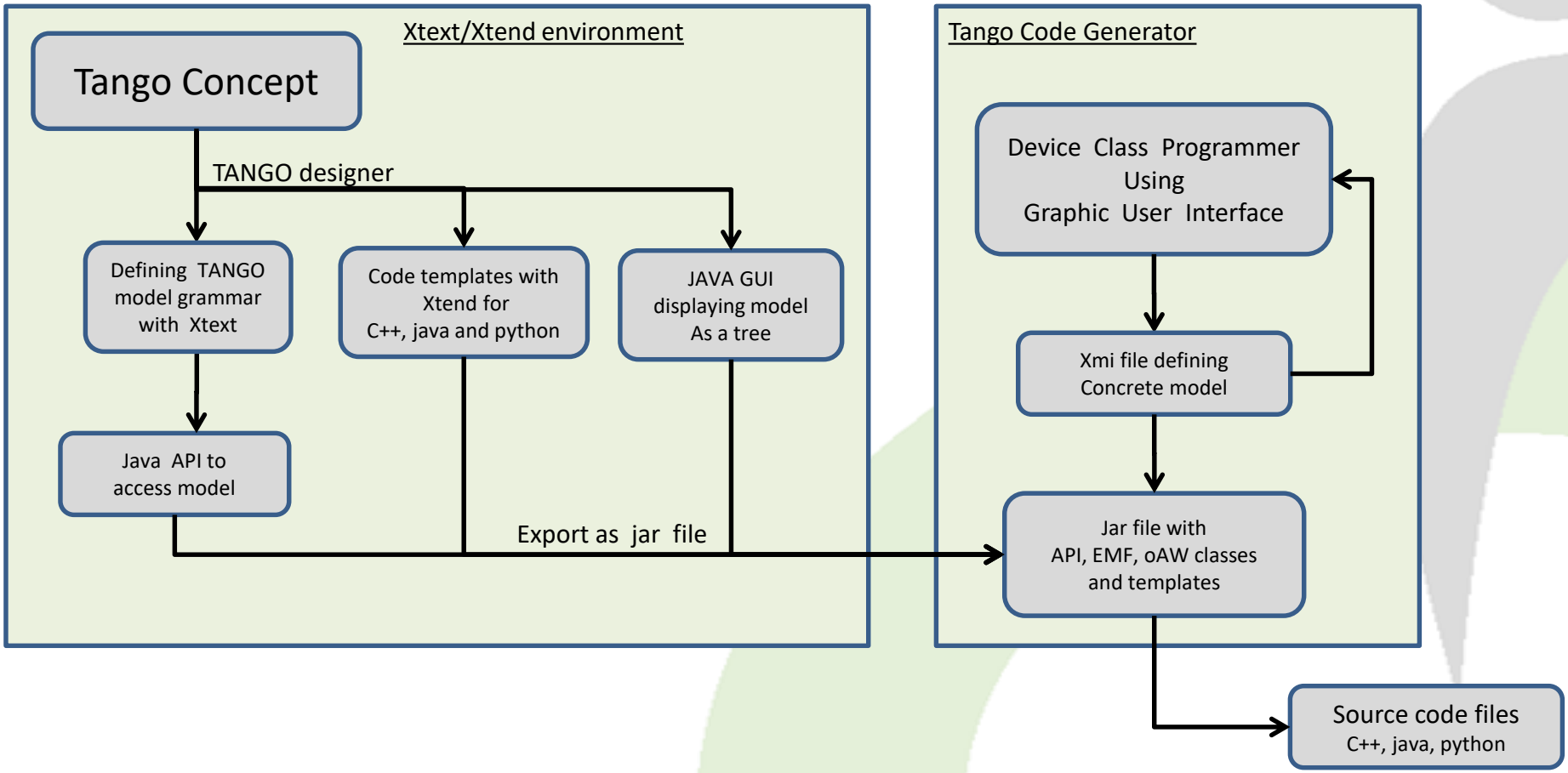
Java Interfaces and **Java classes Implementation**





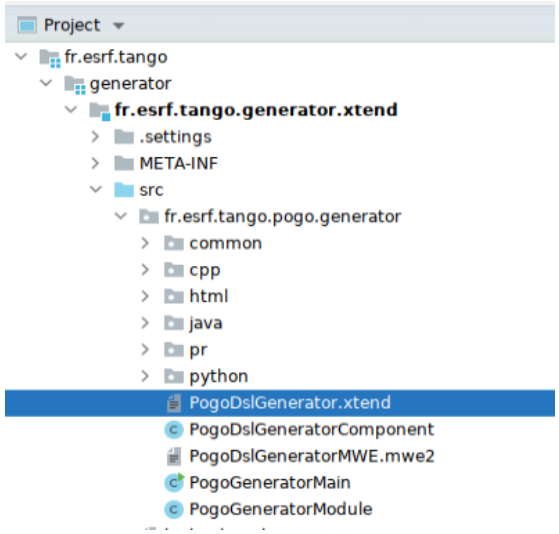
Pogo TANGO code generator

Workflow Principle:



Pogo TANGO code generator

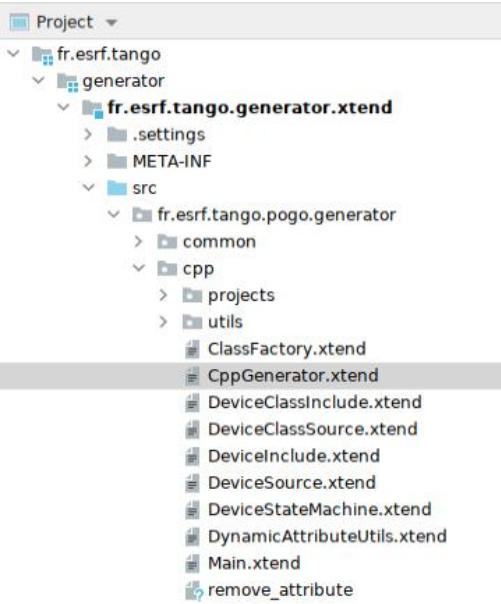
Now you can use these packages to implement code using Xtend.




```
/*
 * generated by Xtext
 */
package fr.esrf.tango.pogo.generator

import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IGenerator
import org.eclipse.xtext.generator.IFileSystemAccess
import com.google.inject.Inject
import fr.esrf.tango.pogo.generator.cpp.CppGenerator
import fr.esrf.tango.pogo.generator.java.JavaGenerator
import fr.esrf.tango.pogo.generator.html.HtmlGenerator
import fr.esrf.tango.pogo.generator.python.PythonDevice
import fr.esrf.tango.pogo.generator.python.PythonDeviceHL

class PogoDslGenerator implements IGenerator {

    @Inject CppGenerator cppDeviceGenerator
    @Inject JavaGenerator javaDeviceGenerator
    @Inject PythonDevice pythonDevice
    @Inject PythonDeviceHL pythonDeviceHL
    @Inject HtmlGenerator htmlGenerator

    override void doGenerate(Resource resource, IFileSystemAccess fsa) {
        cppDeviceGenerator.doGenerate(resource,fsa)
        javaDeviceGenerator.doGenerate(resource,fsa)
        pythonDevice.doGenerate(resource,fsa)
        pythonDeviceHL.doGenerate(resource,fsa)
        htmlGenerator.doGenerate(resource,fsa)
    }
}
```

Pogo TANGO code generator

Now you can use these packages to implement code using Xtend.




```
override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    //
    //  Generate cpp files for PogoDeviceClass objects
    //
    for (cls : resource.allContents.toIterable.filter(typeof(PogoDeviceClass))) {
        //println(cls.description.filestogenerate)
        //println("Lannguage="+cls.description.language)
        if (cls.description.language.toLowerCase.equals("cpp")) {
            //  Code files
            if (cls.description.filestogenerate.contains("Code files")) {
                printTrace("Generating " + cls.deviceIncludeFileName)
                fsa.generateFile(cls.deviceIncludeFileName,      cls.generateDeviceIncludeFile)

                printTrace("Generating " + cls.deviceSourceFileName)
                fsa.generateFile(cls.deviceSourceFileName,       cls.generateDeviceSourceFile)

                printTrace("Generating " + cls.deviceClassIncludeFileName)
                fsa.generateFile(cls.deviceClassIncludeFileName, cls.generateDeviceClassIncludeFile)

                printTrace("Generating " + cls.deviceClassSourceFileName)
                fsa.generateFile(cls.deviceClassSourceFileName,  cls.generateDeviceClassSourceFile)

                printTrace("Generating " + cls.stateMachineFileName)
                fsa.generateFile(cls.stateMachineFileName,       cls.generateStateMachineSourceFile)

                printTrace("Generating ClassFactory.cpp")
                fsa.generateFile("ClassFactory.cpp",             cls.generateClassFactoryFile)

                printTrace("Generating main.cpp")
                fsa.generateFile("main.cpp",                     cls.generateMainFile)

                //  Dynamic attributes if any
                if (cls.dynamicAttributes.size>0 || cls.dynamicCommands.size()>0) {
                    printTrace("Generating " + cls.dynamicAttrUtilsFileName)
                    fsa.generateFile(cls.dynamicAttrUtilsFileName,cls.generateDynamicAttrUtilsFile)
                }
            }
            printTrace("------> " + cls.description.filestogenerate)

            //  Linux Makefile
            if (cls.description.filestogenerate.contains("Makefile")) {
                printTrace("Generating Makefile")
                fsa.generateFile("Makefile",  cls.generateLinuxMakefile)
            }

            //  Linux CMakeLists.txt
            if (cls.description.filestogenerate.contains("CMakeLists")) {
                printTrace("Generating CMakeLists")
                fsa.generateFile("CMakeLists.txt",  cls.generateLinuxCMakeLists)
            }
```

Pogo TANGO code generator

Xtend syntax:

- Method definition:

```
def generateDeviceSourceFile (PogoDeviceClass cls) '''
    - - - -
'''
```

- Everything in this method will be in code generation, except code inside « »
  - Xtend call:

    «IF » «ENDIF», «FOR » «ENDFOR», ….
  - Model data members call:

    #include <«cls.name».h>
  - Xtend or java method call:

    «[param1].[method name]([param2, param3,…])»

    «cls.protectedArea("init_device", "Initialize device", true)»

```
/**
 *  «cls.name» class description:
 *   «cls.description.description.comments("*   ")»
 */
```

Pogo TANGO code generator

```
def generateDeviceSourceFile (PogoDeviceClass cls) '''
   «cls.fileHeader»

   /**
    *  «cls.name» class description:
    *    «cls.description.description.comments("*    ")»
    */

   «cls.commands.commandsTable»

   «cls.attributes.attributesTable»

   namespace «cls.name»_ns
   {
   «cls.protectedArea("namespace_starting", "static initializations", true)»
   «cls.constructors»
   «cls.initDeviceMethod»
   «cls.utilsMethods»
   «cls.attributeMethods»
   «cls.pipeMethods»
   «cls.commandMethods»

   «cls.protectedArea("namespace_ending", "Additional Methods", true)»
   } //          namespace
'''
```

Pogo TANGO code generator

Methods can be written in Xtend

```
def fileHeader (PogoDeviceClass cls) '''
    «cls.protectedArea(".cpp",
        cls.deviceSourceFileHeader+
        "\n\n" +
        "#include <"+cls.name+".h>\n"+
        "#include <"+cls.name+"Class.h>", false)»
'''
```

Ot in java

```
public String protectedArea(PogoDeviceClass cls, String method, String code, boolean comments) {
    if (comments)
            return   openProtectedArea(cls, method)+ "\n" +
                     "//    " + CppStringUtils.comments(code, "    //                ") + "\n\n" +
                     closeProtectedArea(cls, method);
   else
            return  openProtectedArea(cls, method) +
                    code + "\n\n" +
                    closeProtectedArea(cls, method);
}
```

Pogo TANGO code generator

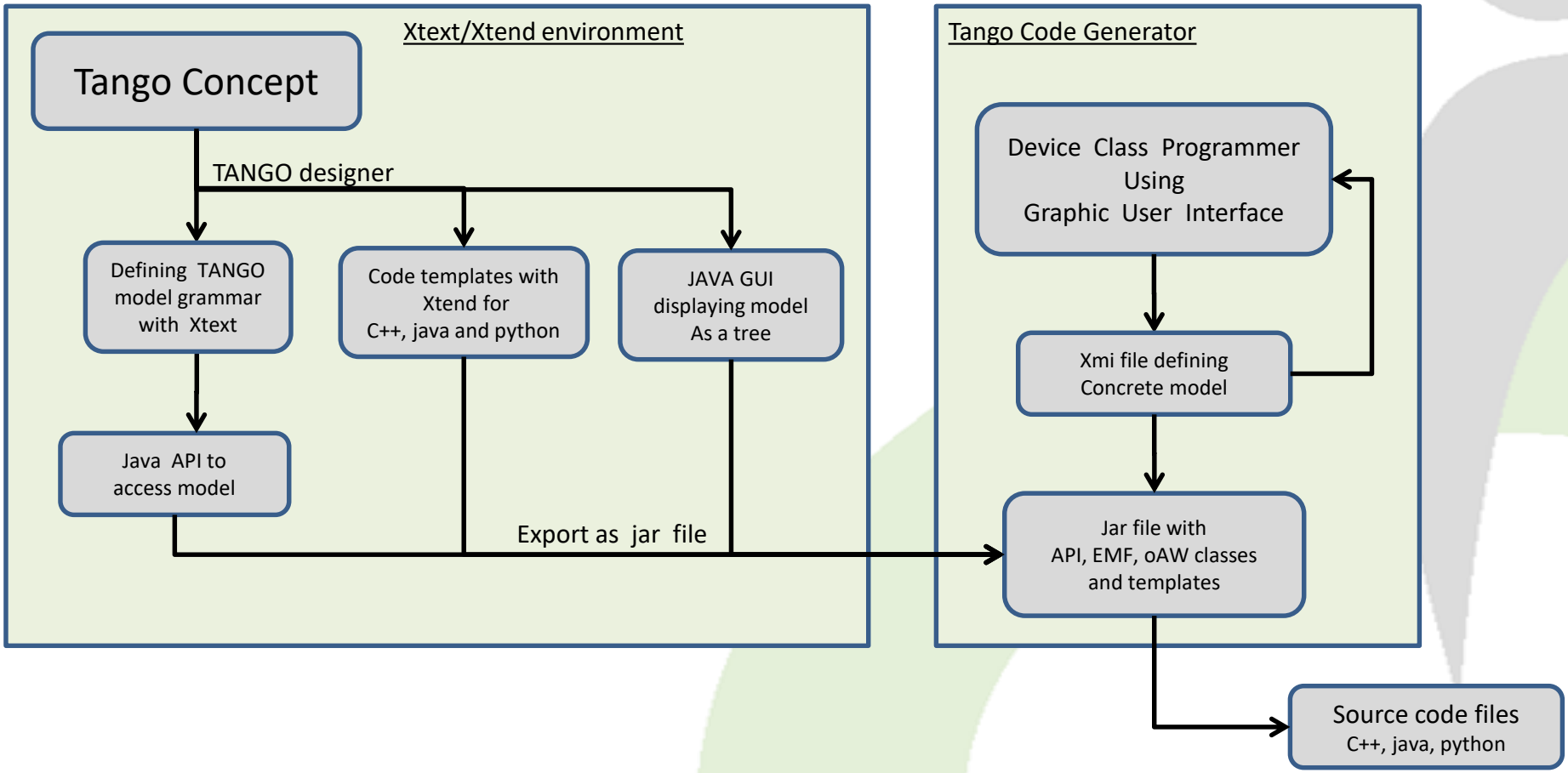
Methods written in Xtend

```
def commandMethods(PogoDeviceClass cls) '''
    «FOR Command command : cls.commands»
        «IF isTrue(command.status.concreteHere)»
            //-------------------------------------------------------
            «command.commandExecutionMethodHeader»
            //-------------------------------------------------------
            «cls.commandExecutionMethod(command)»
        «ENDIF»
    «ENDFOR»
    «IF cls.dynamicCommands.size()>0»
        «FOR Command command : cls.dynamicCommands»
            «IF isTrue(command.status.concreteHere)»
                //-------------------------------------------------------
                «command.commandExecutionMethodHeader»
                //-------------------------------------------------------
                «cls.commandExecutionMethod(command)»
            «ENDIF»
        «ENDFOR»
    «ENDIF»
'''
```

Xtend uses special cotes: **«** Xtend code **»**
Under Eclipse these cotes are [**Ctrl** + **Shift** + ‘**<**‘ or ‘**>**’]
It seems not possible with Intelli JIdea except with copy/paste

Pogo TANGO code generator

Workflow Principle:



Pogo TANGO code generator

Java GUI:

The java GUI, loads a xmi file as a **PogoDeviceClass** object (OAWutils.java) as defined in PogoDsl.xtext file

```
PogoDeviceClass:
   "deviceclass" name=ID (isAbstract?="abstract")? ("extends" baseClass=[PogoDeviceClass])?
  "{"
       pogoRevision = ID                          // To check Pogo revision when xmi has been created
       institute = ID                             // For future specific features
       "description:"                             // Class information
                  (description=ClassDescription)
      "classProperties:"                          // Class property list
                  (classProperties+=Property)*
      - - - - - - -
```

And convert it in a **DeviceClass** object to be used during design.

When Generate option is clicked, a new **DeviceClass** object is build from the tree, converted in **PogoDeviceClass** object passed to OAWutils.java file to be saved in a xmi file (and generate code if requested).

For compatibility reasons, few methods modify the xmi file content just before load.
And just before code generation

Pogo TANGO code generator

Remarks about Protected Regions(PR):

The PRs are fully managed the lib package.

Each PR needs to have a **unique** name. It is build using class name, method name and/or usage.

i.e.

```
/*----- PROTECTED REGION ID(Starter::reset_statistics) ENABLED START -----*/
            Bla bla bla….
/*----- PROTECTED REGION END -----*/     //            Starter::reset_statistics
```

But: if a command name has changed, the execution method name will change too.

Consequently, the PR name change too.

Xtend cannot recognise it and the protected code is lost.

To avoid it, when an object name change, the old and new names are added in a list to get code from old PR before code generation and reinjected in PR with new name after.

TANGO

Connecting things together

Thank you!

Questions?

http://www.tango-controls.org/