Porting a Java-based Brain Simulation Software to C++

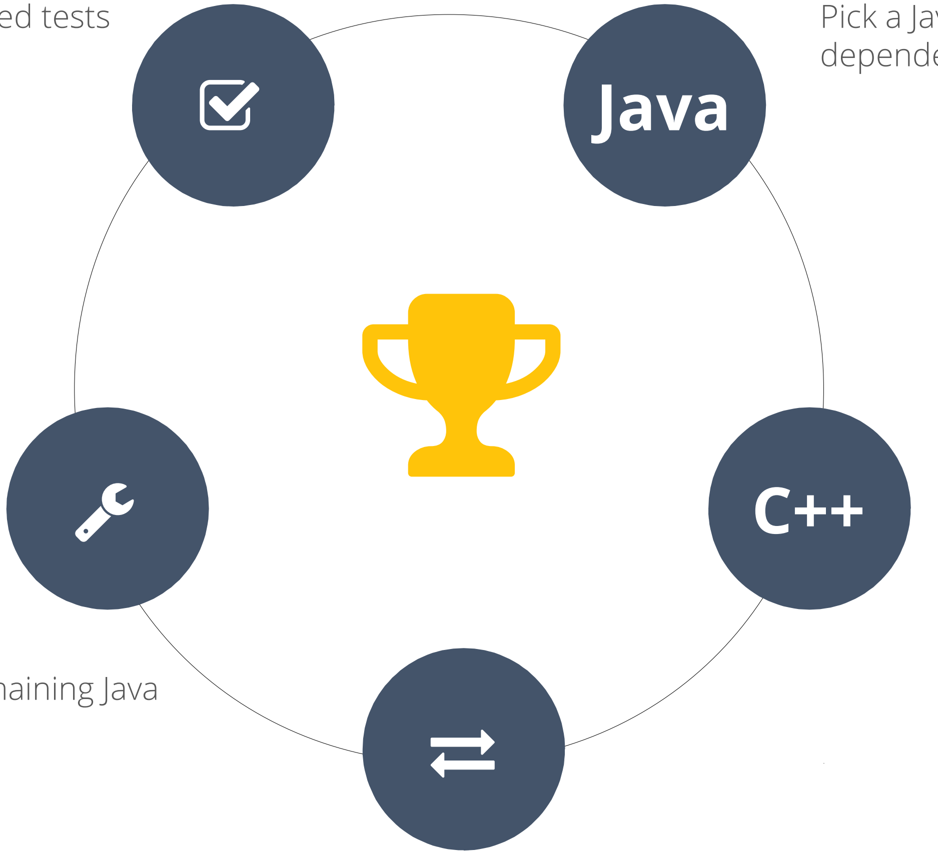
Lukas Johannes Breitwieser
https://at.linkedin.com/in/lukasbreitwieser

Simulation of a Self-organizing Neural Network Using Axonal Growth Rules

Roman Bauer

The decision to port Cx3D to **C++** was driven by our goal to **simulate deeper & richer structures**.

ITERATIVE PORTING WORKFLOW



Step 1

Pick a Java class with few dependencies

Step 2

Translate the Java code into C++

Step 3

Write code to enable communication between Java and C++

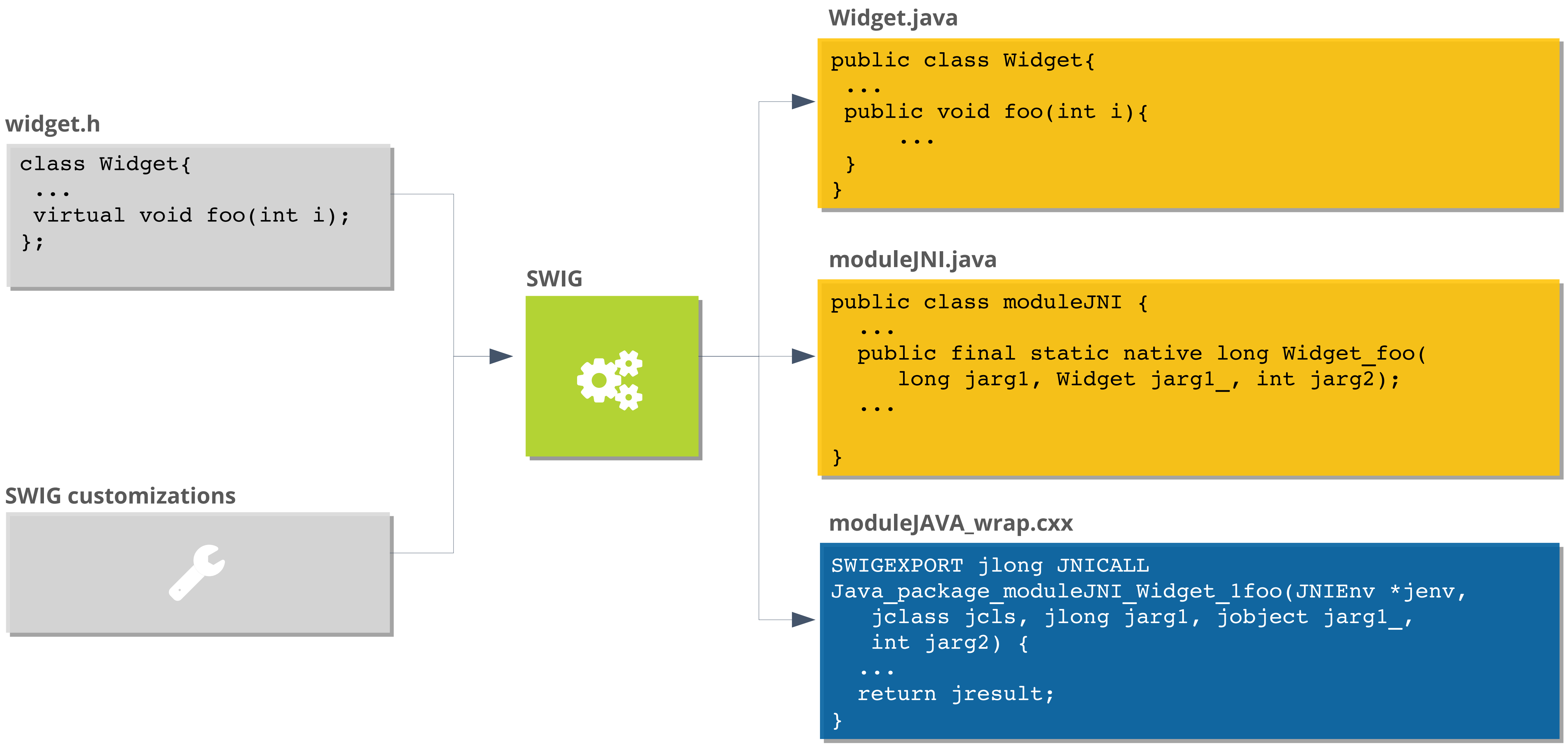
Step 4

Refactor remaining Java application

Step 5

Run automated tests

COMMUNICATION BETWEEN JAVA AND C++



widget.h

```
class Widget{
 ...
 virtual void foo(int i);
};
```

SWIG customizations

SWIG

Widget.java

```
public class Widget{
 ...
 public void foo(int i){
     ...
 }
}
```

moduleJNI.java

```
public class moduleJNI {
  ...
  public final static native long Widget_foo(
     long jarg1, Widget jarg1_, int jarg2);
  ...

}
```

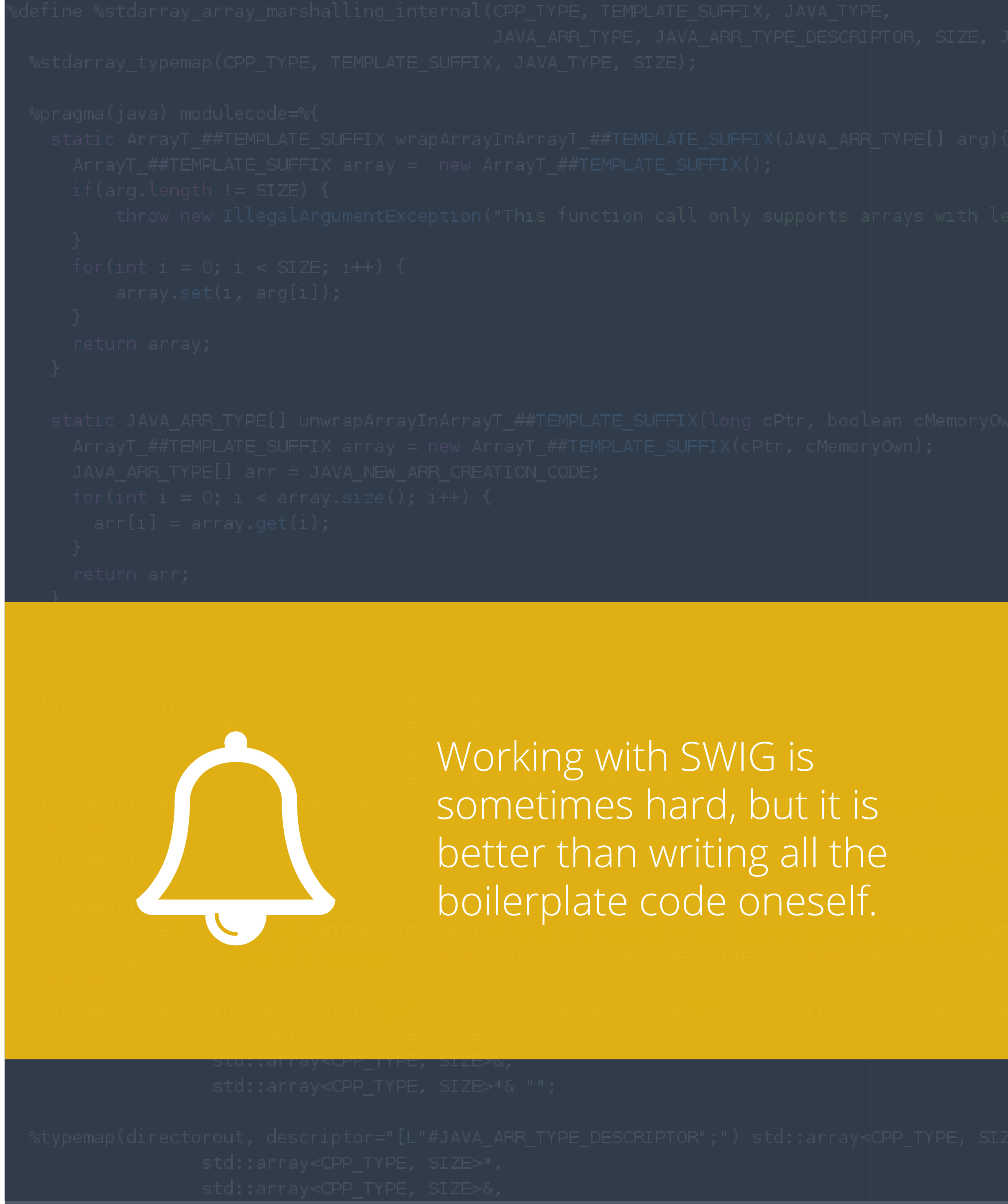
moduleJAVA_wrap.cxx

```
SWIGEXPORT jlong JNICALL
Java_package_moduleJNI_Widget_1foo(JNIEnv *jenv,
   jclass jcls, jlong jarg1, jobject jarg1_,
   int jarg2) {
  ...
  return jresult;
}
```

SWIG CUSTOMIZATION

- Rules for type conversions and type modifications
  e.g. function with parameter **`const std::array<std::shared_ptr<Rational>, 3>&`** that should translate into **`Rational[]`** on the Java side

- Two-way-communication
  e.g. Java defined callback that is passed on to the native implementation and invoked from there.



```
%define %stdarray_array_marshalling_internal(CPP_TYPE, TEMPLATE_SUFFIX, JAVA_TYPE,
                                             JAVA_ARR_TYPE, JAVA_ARR_TYPE_DESCRIPTOR, SIZE, J
  %stdarray_typemap(CPP_TYPE, TEMPLATE_SUFFIX, JAVA_TYPE, SIZE);

  %pragma(java) modulecode=%{
    static ArrayT_##TEMPLATE_SUFFIX wrapArrayInArrayT_##TEMPLATE_SUFFIX(JAVA_ARR_TYPE[] arg){
      ArrayT_##TEMPLATE_SUFFIX array =  new ArrayT_##TEMPLATE_SUFFIX();
      if(arg.length != SIZE) {
          throw new IllegalArgumentException("This function call only supports arrays with le
      }
      for(int i = 0; i < SIZE; i++) {
          array.set(i, arg[i]);
      }
      return array;
    }

    static JAVA_ARR_TYPE[] unwrapArrayInArrayT_##TEMPLATE_SUFFIX(long cPtr, boolean cMemoryOw
      ArrayT_##TEMPLATE_SUFFIX array = new ArrayT_##TEMPLATE_SUFFIX(cPtr, cMemoryOwn);
      JAVA_ARR_TYPE[] arr = JAVA_NEW_ARR_CREATION_CODE;
      for(int i = 0; i < array.size(); i++) {
        arr[i] = array.get(i);
      }
      return arr;
```

```
                 std::array<CPP_TYPE, SIZE>*& "";

  %typemap(directorout, descriptor="[L"#JAVA_ARR_TYPE_DESCRIPTOR";") std::array<CPP_TYPE, SIZ
                std::array<CPP_TYPE, SIZE>*,
                std::array<CPP_TYPE, SIZE>&,
```

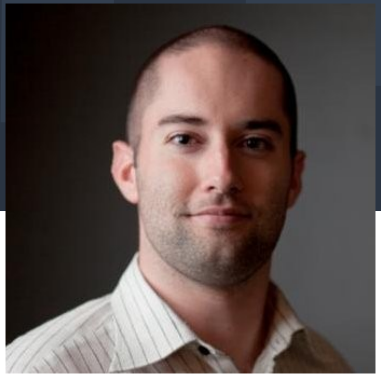
Working with SWIG is sometimes hard, but it is better than writing all the boilerplate code oneself.

MANY THANKS TO MY SUPERVISORS / SUPPORTERS

Alberto Di Meglio

Fons Rademakers

Marco Manca


Roman Bauer