Wrapping Legacy codes for Grid-Based Applications

Yan Huang, David Walker, Ian Taylor and Robert Davies

Cardiff University

Background

- Java becomes a powerful and popular language for developing Grid architecture and Grid-based e-Science applications.
- A large body of existing high quality, validated code is written in non-Java language.
- It is daunting and expensive to covert them to Grid-enabled, Java-based services.

Aim

- Want to automatically componentize legacy software.
- Especially large libraries of scientific and numerical software.
- Typical approach is to “wrap” each piece of code for use as a component.

JACAW

- JAva-C Automatic Wrapper
- JACAW automatically generates the JNI interface for making C native method calls from Java.
- Uses the C header files.
- Complete source code is not needed.

JACAW Example

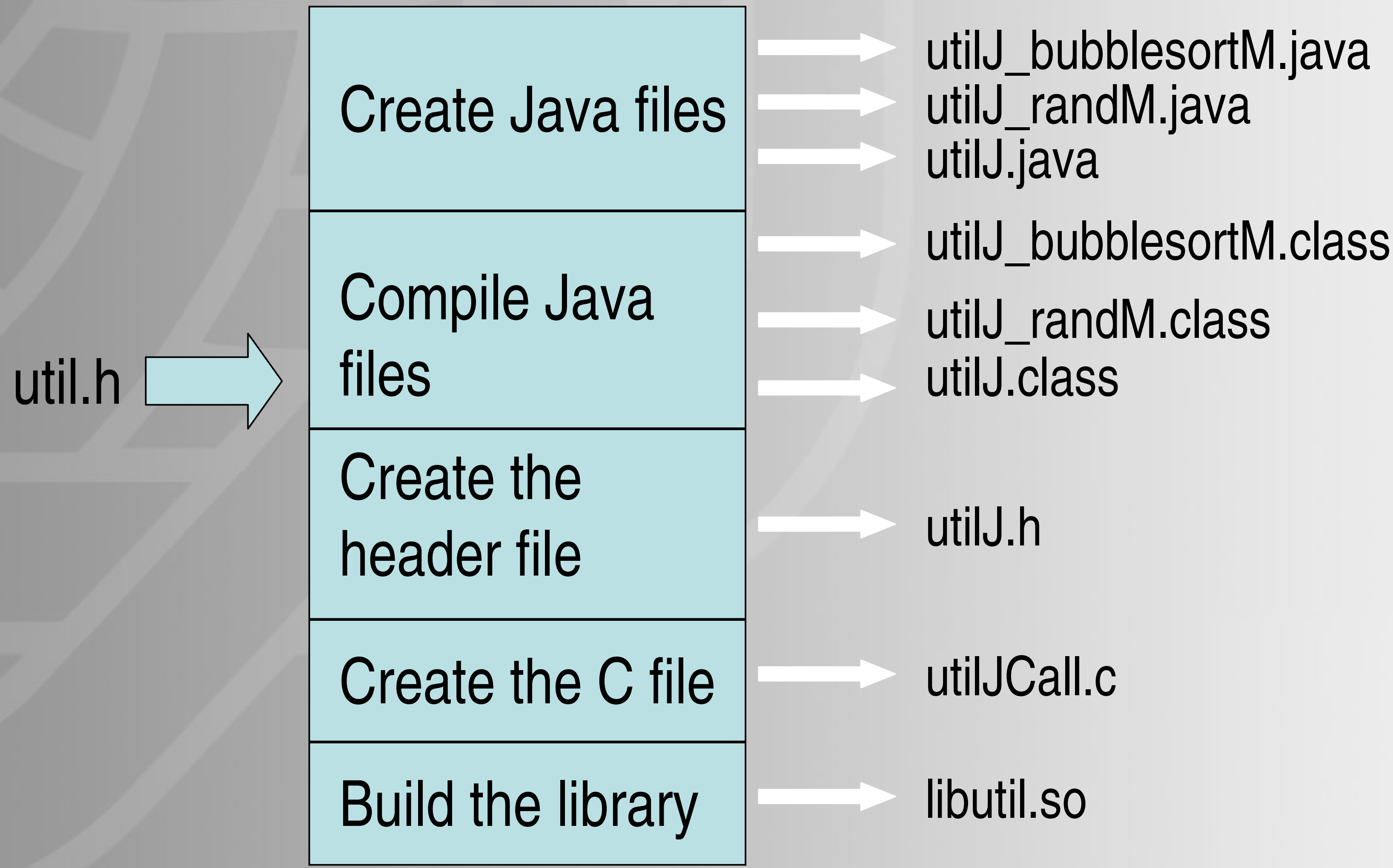
- Suppose the header file util.h contains:

  signed int rand ();

  void bubblesort (int, int[]);

- JACAW builds class files for the two routines and the JNI interface code for them.

What JACAW Does



```java
public class utilJ_bubblesortM{
    boolean needCopy = true;
    int arg0;
    int[] arg1;

    public utilJ_bubblesortM(int arg0, int[] arg1){
        this.arg0 = arg0;        this.arg1 = arg1;
    }
    public void nativeCall(){
        utilJ.bubblesort(this.arg0, this.arg1, this.needCopy);
    }

    public int getArg0(){  return this.arg0;  }
    public int[] getArg1(){  return this.arg1;  }
    public void setNeedCopy(Boolean needCopy){
            this.needCopy = needCopy;
    }
}
```

utilJ.java

```
/* THIS CODE IS AUTOMATICALLY CREATED BY JACAW */
public class utilJ{
    public native static int rand();
    public native static void bubblesort(int arg0, int[] arg1, boolean needCopy);
    static {
        System.loadLibrary("util");
    }
}
```

utilJcall.c

```
#include <jni.h>
#include "/home/scmyh/test/JACAW/try1/util/util.h"
JNIEXPORT void JNICALL Java_utilJ_bubblesort(JNIEnv * env,
    jclass cls, jint arg0, jintArray arg1, jboolean needCopy) {
    /* This is a native call for method: void bubblesort(int, int [])*/
    jint* arg1P;
    int mode;
    arg1P = (*env)->GetIntArrayElements(env, arg1, NULL);
    bubblesort(arg0, arg1P);
    if (needCopy == JNI_TRUE) mode = 0;
    else mode = JNI_ABORT;
    (*env)->ReleaseIntArrayElements(env, arg1, arg1P, mode);
}
```

How To Call a C Routine

- Initiate the class corresponding to the wrapped routine.
- Call setNeedCopy() to set needCopy variable.
- Call nativeCall().
- Call getReturnValue() to get return value.
- Call getArgX() to retrieve the argument that may have been modified.

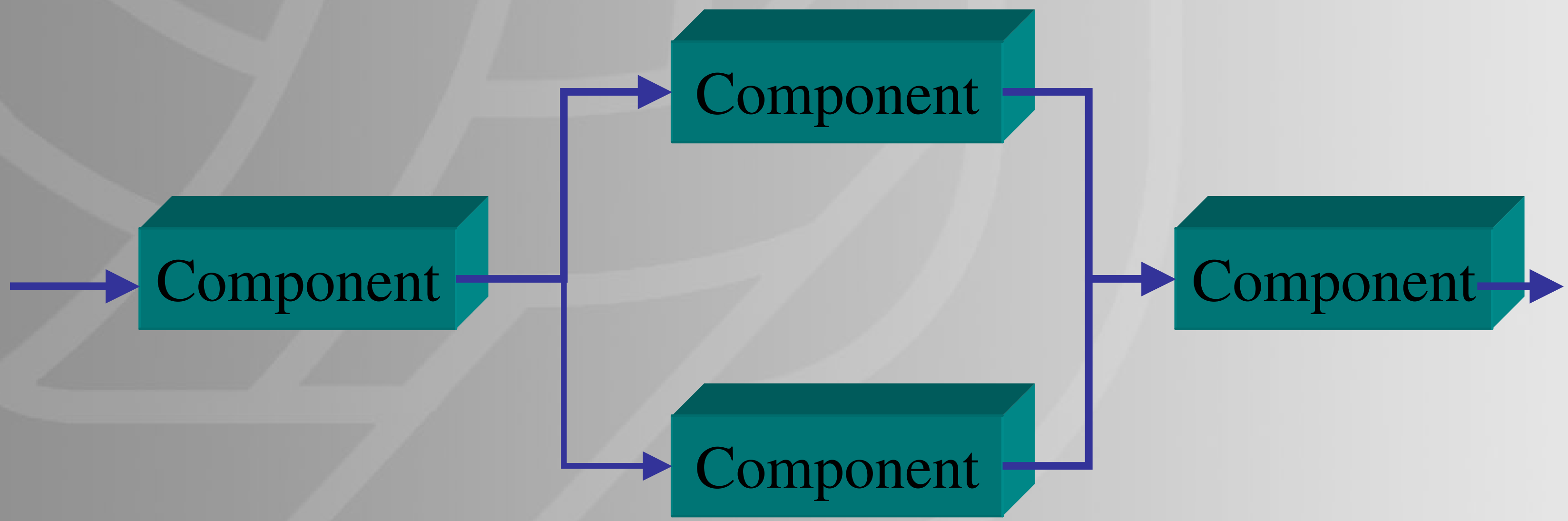
Example

```
int[] v = new int[100];
int len = 100;
utilJ_randM rand = new utilJ_randM();
for ( int i=0;  i<len;  i++ ) {
        rand.nativeCall();
        v[i]=rand.getReturnValue();
}
utilJ_bubblesortM sort = new utilJ_bubblesortM(len, v);
sort.nativeCall();
v = sort.getArg1();
```

Triana

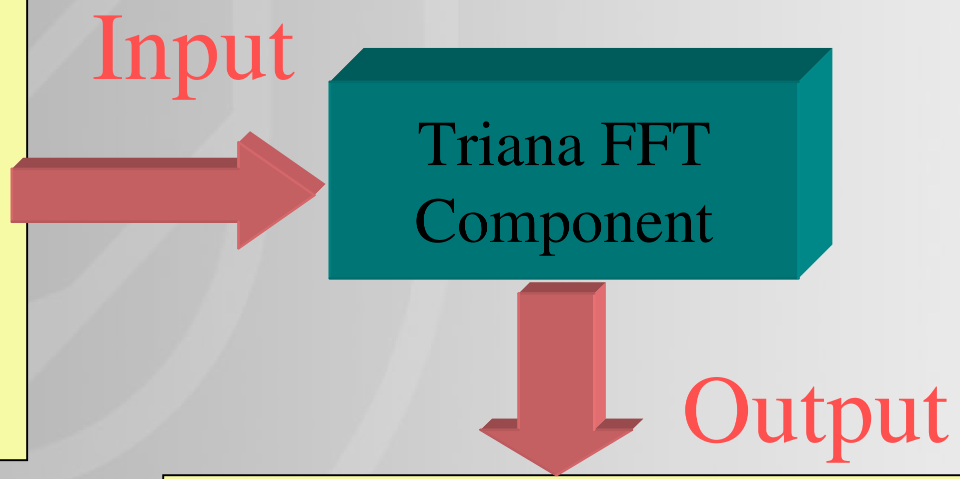
- Triana is a graphical programming environment that allows users to create Java-based applications by dragging and dropping the components and then connecting them together.

Triana Composite Application



An Example of Triana Component

```
class SampleSet …{
  public double data[];
  public double samplingFrequency;
  …
  get and set methods.
}
```



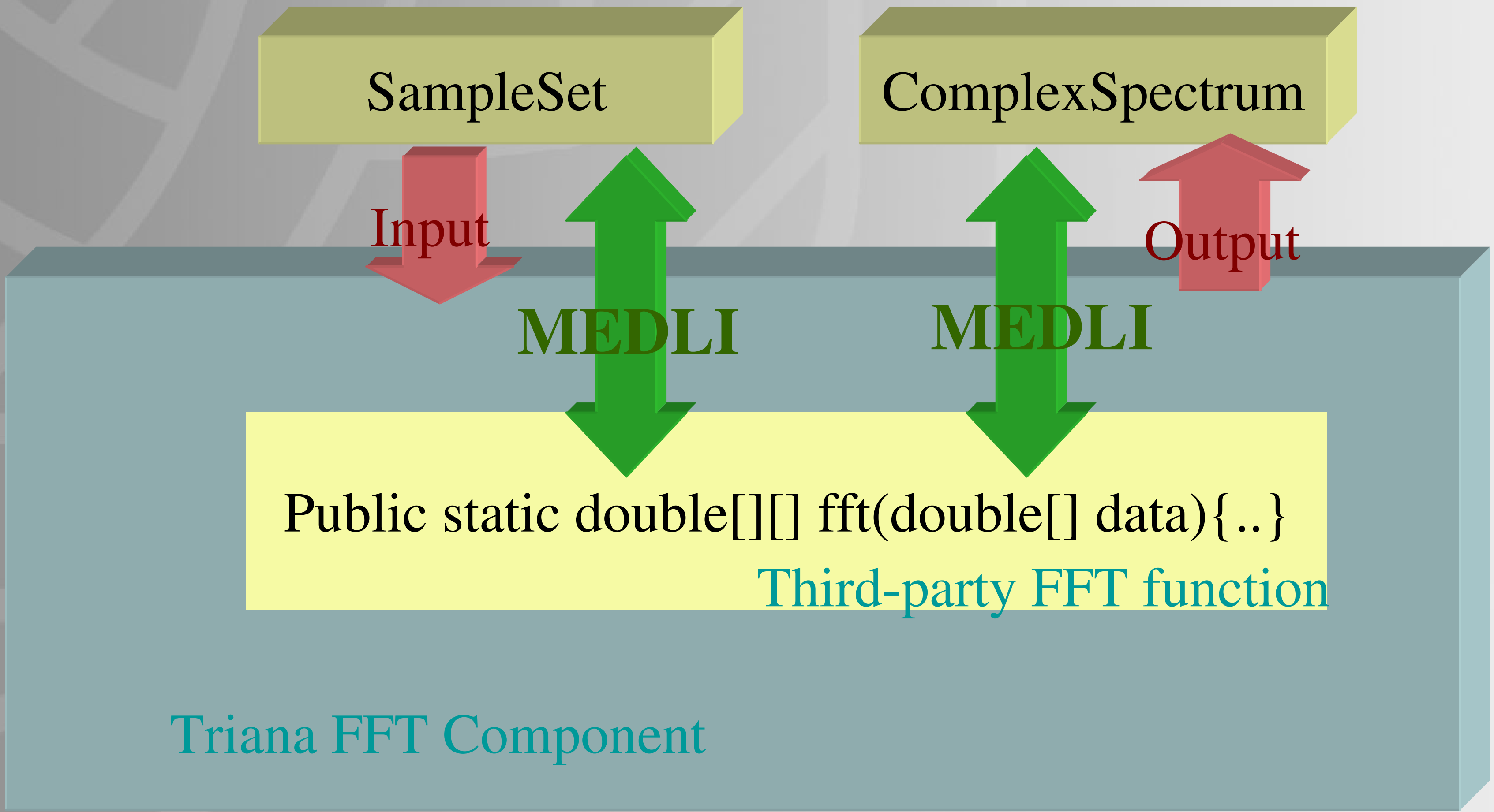

```
class ComplexSpectrum …{
  public double real[];
  public double imag[];
  public double samplingFrequency;
  …
  get and set methods.
}
```

Triana components use pre-defined data types.

MEDLI

- A Graphical environment for mediating the conversion of data types between the classes of a Java application and the inputs of a third-party Java software.

Triana And MEDLI Together



MEDLI Mediation Wizard

- Allows user graphically map some or all of the public variables in a predefined object to the input of a third-party target routine.
- Allows user graphically map the output of a third-party target routine to a pre-defined object.
- Allows some data in the input object directly pass to the output object.

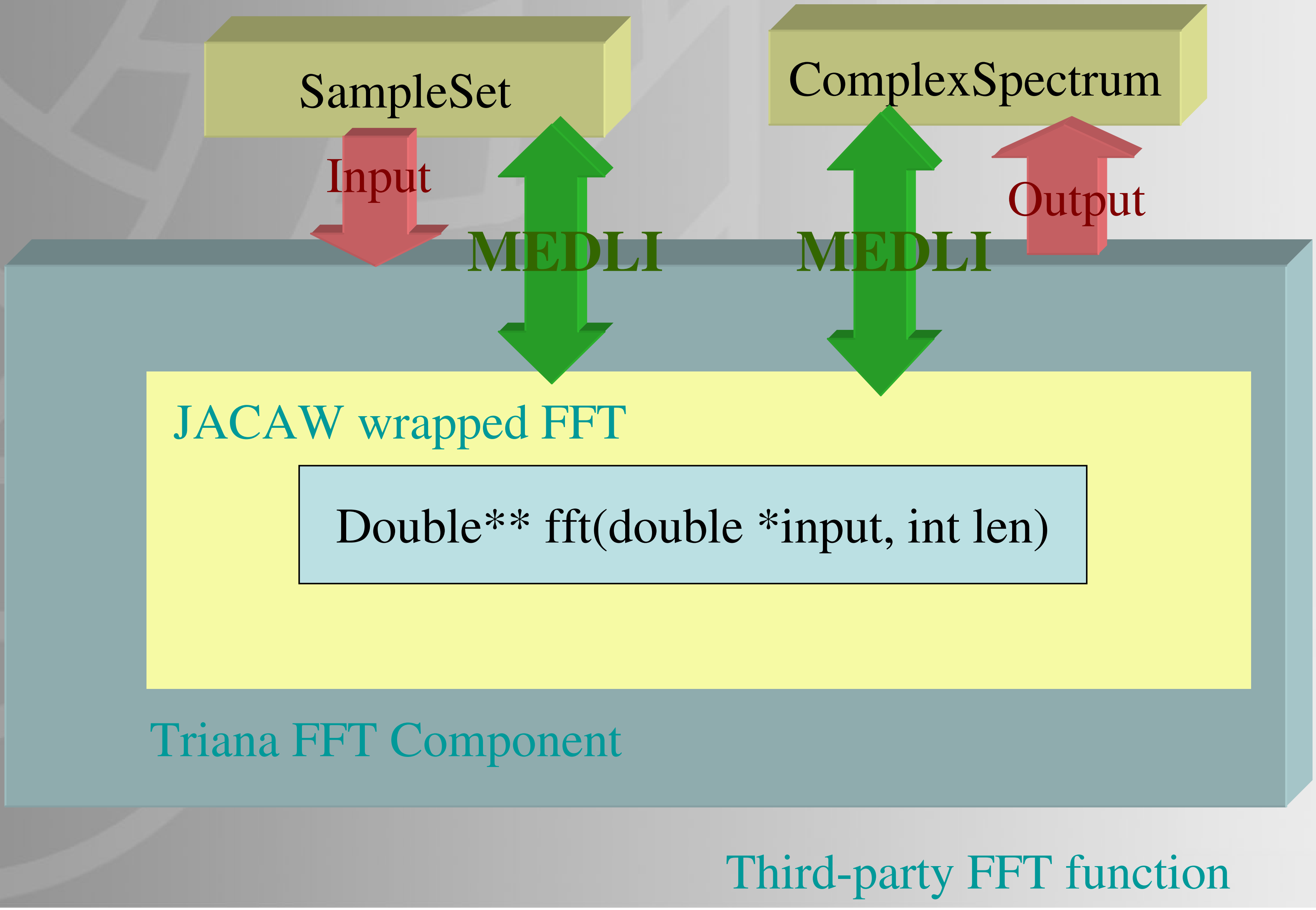
MEDLI Generated Code

```
Public class MEDLI_FFT{
    public static ComplexSpectrum FFT(SampleSet input){
        double[][] output = FFT.fft(input input.getData());
        ComplexSpectrum retVal = new ComplexSpectrum();
        retVal.setReal(output[0]);
        retVal.serImag(output[1]);
        …
        Return retVal;
}
```

Native Calls and MEDLI

- By integrating JACAW, MEDLI supports the mediation of data between Java and C.
- And the c-implemented routines can be easily wrapped as a Triana-based components.

A New Picture



```
public class FFTJ_fftM{
    double[] arg0;    int arg1;
    double[][] retVal=null;
    boolean needCopy = true;

    public FFTJ_fftM(double[][] arg0, int[] arg1){
        this.arg0 = arg0;        this.arg1 = arg1;
    }
    public void nativeCall(){
        this.retVal=FFTJ.fft(this.arg0, this.arg1, this.needCopy);
    }
    public double[][] getArg0(){  return this.arg0;  }
    public int getArg1(){  return this.arg1;  }
    public double[][] getReturnValue{ return this.retVal;};
    public void setNeedCopy(Boolean needCopy){
            this.needCopy = needCopy;
    }
}
```

MEDLI Generated Code

```
Public class MEDLI_FFT{
   public static ComplexSpectrum FFT(SampleSet input){
        FFTJ_fftM fft = new FFTJ_ffftM(input.getData(),
                input.getData().length);
        fft.nativeCall();
        double[][] output = fft.getReturnValue();
        ComplexSpectrum retVal = new ComplexSpectrum();
        retVal.setReal(output[0]);
        retVal.serImag(output[1]);
        …
        Return retVal;
}
```

Conclusion

- JACAW provides a fast and convenient way of enabling legacy C routines to be called from Java applications.
- Triana, MEDLI and JACAW together presents an environment allowing legacy C routines be presented as Triana-based components to composite Java applications.