TECHNICAL REPORT
IGE–320

A USER GUIDE FOR JARGON VERSION5

A. Hébert

Institut de génie nucléaire
Département de génie mécanique
École Polytechnique de Montréal

March 9, 2012

Contents

1 Introduction

The current version of Jargon uses calculation operators borrowed from DRAGON[1] and DONJON[2]. The remote dispatching and asychroneous calculation capabilities of Jargon are described in Ref. 3. A user guide for Jargon is available in Ref. 4.

Jargon Version5 is built on top of Ganlib Version5.[5] This version of Jargon is therefore 64-bit clean and its ANSI-C and Fortran components are ISO-compliant.

Jargon can be executed in multiple ways. The simplest approach is to use BeanShell scripts[5] to write complete data files. It is also possible to write a library of Computational schemes in Java or in Cle-2000 and to call them from user-defined BeanShell scripts. Finally, we always have the possibility to use graphical user interfaces (GUI) instead of BeanShell scripts[6].

2 Examples of BeanShell scripts

The following examples are implemented as BeanShell scripts using classes from the Jargon framework. Jargon contains two demo examples of Computational Schemes: SimpleCell and SimpleCompo. The complete JavaDoc documentation is available in Ref. 4.

2.1 The IAEA-3D Diffusion Benchmark

```
import jargon.*;

myMacro = new Macrolib("myMacro", 2, 1, 1);
// mixture 1:
M1 = new Composition("M1", "medium");
myMacro.setDiff(M1, new float[] {1.5f, 0.4f});
myMacro.setTotal(M1, new float[] {3.0e-2f, 8.0e-2f});
myMacro.setNuSigf(M1, new float[] {0.0f, 0.135f});
myMacro.setChi(M1, new float[] {1.0f, 0.0f});
myMacro.setHFactor(M1, new float[] {0.0f, 0.135f});
myMacro.setScat(M1, 0, 1, new float[] {0.0f}, 1, 1);
myMacro.setScat(M1, 0, 2, new float[] {0.0f, 0.02f}, 2, 2);
// mixture 2:
M2 = new Composition("M2", "medium");
myMacro.setDiff(M2, new float[] {1.5f, 0.4f});
myMacro.setTotal(M2, new float[] {3.0e-2f, 8.5e-2f});
myMacro.setNuSigf(M2, new float[] {0.0f, 0.135f});
myMacro.setChi(M2, new float[] {1.0f, 0.0f});
myMacro.setHFactor(M2, new float[] {0.0f, 0.135f});
myMacro.setScat(M2, 0, 1, new float[] {0.0f}, 1, 1);
myMacro.setScat(M2, 0, 2, new float[] {0.0f, 0.02f}, 2, 2);
// mixture 3:
M3 = new Composition("M3", "medium");
myMacro.setDiff(M3, new float[] {1.5f, 0.4f});
myMacro.setTotal(M3, new float[] {3.0e-2f, 0.13f});
myMacro.setNuSigf(M3, new float[] {0.0f, 0.135f});
myMacro.setChi(M3, new float[] {1.0f, 0.0f});
myMacro.setHFactor(M3, new float[] {0.0f, 0.135f});
myMacro.setScat(M3, 0, 1, new float[] {0.0f}, 1, 1);
myMacro.setScat(M3, 0, 2, new float[] {0.0f, 0.02f}, 2, 2);
// mixture 4:
M4 = new Composition("M4", "medium");
myMacro.setDiff(M4, new float[] {2.0f, 0.3f});
myMacro.setTotal(M4, new float[] {4.0e-2f, 1.0e-2f});
myMacro.setScat(M4, 0, 1, new float[] {0.0f}, 1, 1);
myMacro.setScat(M4, 0, 2, new float[] {0.0f, 0.04f}, 2, 2);
// mixture 5:
M5 = new Composition("M5", "medium");
myMacro.setDiff(M5, new float[] {2.0f, 0.3f});
myMacro.setTotal(M5, new float[] {4.0e-2f, 5.5e-2f});
myMacro.setScat(M5, 0, 1, new float[] {0.0f}, 1, 1);
myMacro.setScat(M5, 0, 2, new float[] {0.0f, 0.04f}, 2, 2);

iaea3d = new Gigogne("iaea3d", "CAR3D", new int[]{9, 9, 4});
iaea3d.bc = new String[][] {{"X-","DIAG"},{"X+","VOID"},{"Y-","SYME"},
                            {"Y+","DIAG"},{"Z-","VOID"},{"Z+","VOID"},};
o = null;
iaea3d.media = new Composition[]{ M4, M4, M4, M4, M4, M4, M4, M4, M4,
                                      M4, M4, M4, M4, M4, M4, M4, M4,
                                          M4, M4, M4, M4, M4, M4, M4,
                                              M4, M4, M4, M4, M4, M4,
                                                  M4, M4, M4, M4,  o,
                                                      M4, M4, M4,  o,
                                                          M4,  o,  o,
                                                               o,  o,
                                                                   o,
                                  M3, M2, M2, M2, M3, M2, M2, M1, M4,
                                      M2, M2, M2, M2, M2, M2, M1, M4,
                                          M2, M2, M2, M2, M1, M1, M4,
                                              M2, M2, M2, M1, M4, M4,
                                                  M3, M1, M1, M4,  o,
                                                      M1, M4, M4,  o,
                                                          M4,  o,  o,
                                                               o,  o,
                                                                   o,
                                  M3, M2, M2, M2, M3, M2, M2, M1, M4,
                                      M2, M2, M2, M2, M2, M2, M1, M4,
                                          M3, M2, M2, M2, M1, M1, M4,
                                              M2, M2, M2, M1, M4, M4,
                                                  M3, M1, M1, M4,  o,
                                                      M1, M4, M4,  o,
                                                          M4,  o,  o,
                                                               o,  o,
                                                                   o,
                                  M5, M4, M4, M4, M5, M4, M4, M4, M4,
                                      M4, M4, M4, M4, M4, M4, M4, M4,
                                          M5, M4, M4, M4, M4, M4, M4,
                                              M4, M4, M4, M4, M4, M4,
                                                  M5, M4, M4, M4,  o,
                                                      M4, M4, M4,  o,
                                                          M4,  o,  o,
                                                               o,  o,
                                                                   o};
iaea3d.meshx = new float[]{0.0f, 20.0f, 40.0f, 60.0f, 80.0f, 100.0f,
                           120.0f, 140.0f, 160.0f, 180.0f};
iaea3d.meshz = new float[]{0.0f, 20.0f, 280.0f, 360.0f, 380.0f};
iaea3d.splitz = new int[] {1, 2, 1, 1};
iaea3d.text(myMacro);

myScheme = new SimpleCell("myScheme");
myScheme.setGigogne(iaea3d);
myScheme.setMacrolib(myMacro);
myScheme.setSolutionType("Trivac");
myScheme.setMaxr(500);
myScheme.setBase("DUAL", new int[]{3, 3});
myScheme.run();

source("assertS.bsh");
assertS(myScheme.operatorOut.lcmObj, "K-EFFECTIVE", 1, 1.028981f);
```

Script `assertS.bsh` is written:

```
assertS(lcmobj,key,ipos,refvalue) {
  record=lcmobj.get(key);
  myvalue=record[ipos-1];
  if (Math.abs(myvalue - refvalue) > 1.0e-4 * Math.abs(refvalue)) {
    System.out.println("============== error detected =================");
    System.out.println("Reference=" + refvalue + " Calculated=" + myvalue);
    throw new RuntimeException("exception in assertS");
  } else {
    System.out.println("Test successful");
  }
}
```

2.2 A Wigner-Seitz lattice cell with Macrolib

```
import jargon.*;

myMacro = new Macrolib("myMacro", 1, 1, 1);
// mixture 1:
fuel = new Composition("fuel", "medium");
myMacro.setTotal(fuel, new float[] {0.36522f});
myMacro.setNuSigf(fuel, new float[] {0.05564f});
myMacro.setChi(fuel, new float[] {1.0f});
myMacro.setScat(fuel, 0, 1, new float[] {0.3234f}, 1, 1);
// mixture 2:
clad = new Composition("clad", "medium");
myMacro.setTotal(clad, new float[] {0.4029f});
myMacro.setScat(clad, 0, 1, new float[] {0.4000f}, 1, 1);
// mixture 3:
water = new Composition("water", "medium");
myMacro.setTotal(water, new float[] {0.3683f});
myMacro.setScat(water, 0, 1, new float[] {0.3661f}, 1, 1);

myGeom = new Gigogne("myGeom", "CARCEL", new int[]{2});
myGeom.bc = new String[][] {{"X-","REFL"},{"X+","REFL"},{"Y-","REFL"},
                            {"Y+","REFL"}};
myGeom.meshx = new float[] {0.0f, 3.6f};
myGeom.meshy = myGeom.meshx;
myGeom.radius = new float[] {0.0f, 0.829f, 1.029f};
myGeom.media = new Composition[] {fuel, clad, water};

myScheme = new SimpleCell("myScheme");
myScheme.setGigogne(myGeom);
myScheme.setMacrolib(myMacro);
myScheme.setOption("K");
myScheme.run();

source("assertS.bsh");
assertS(myScheme.operatorOut.lcmObj, "K-EFFECTIVE", 1, 1.0479352f);
```

2.3 A Double-Heterogeneity lattice case with Macrolib

```
import jargon.*;

myMacro = new Macrolib("myMacro", 1, 1, 1);
// mixture 1:
water = new Composition("water", "medium");
myMacro.setTotal(water, new float[] {0.3683f});
myMacro.setScat(water, 0, 1, new float[] {0.3661f}, 1, 1);
// mixture 2:
fuel = new Composition("fuel", "medium");
myMacro.setTotal(fuel, new float[] {0.36522f});
myMacro.setNuSigf(fuel, new float[] {0.05564f});
myMacro.setChi(fuel, new float[] {1.0f});
myMacro.setScat(fuel, 0, 1, new float[] {0.3234f}, 1, 1);
```

```
// mixture 3:
clad1 = new Composition("clad1", "medium");
myMacro.setTotal(clad1, new float[] {0.8453f});
myMacro.setScat(clad1, 0, 1, new float[] {0.5216f}, 1, 1);
// mixture 4:
clad2 = new Composition("clad", "medium");
myMacro.setTotal(clad2, new float[] {0.3683f});
myMacro.setScat(clad2, 0, 1, new float[] {0.0f}, 1, 1);
// double heterogeneity mixtures 5 and 6:
fuelGrain1 = new Composition("fuelgrain1", "medium");
fuelGrain2 = new Composition("fuelgrain2", "medium");

cote = 1.262082f;
lame = 1.322082f;
C1 = new Gigogne("C1", "CARCEL", new int[]{0});
C1.meshx = new float[]{0.0f, cote};
C1.meshy = C1.meshx;
C1.media = new Composition[]{clad2};

C2 = new Gigogne("C2", "CARCEL", new int[]{3});
C2.meshx = new float[]{0.0f, cote};
C2.meshy = C2.meshx;
C2.radius = new float[]{0.0f, 3.25296e-01f, 4.60039e-01f, 5.6343e-01f};
C2.media = new Composition[]{fuelGrain1, clad1, clad1, water};

C3 = new Gigogne("C3", "CARCEL", new int[]{1});
C3.meshx = new float[]{0.0f, cote};
C3.meshy = C3.meshx;
C3.radius = new float[]{0.0f, 4.1266e-01f};
C3.media = new Composition[]{fuelGrain2, water};

C4 = new Gigogne("C4", "CARCEL", new int[]{1});
C4.meshx = new float[]{0.0f, lame};
C4.meshy = new float[]{0.0f, cote};
C4.radius = new float[]{0.0f, 4.1266e-01f};
C4.media = new Composition[]{fuelGrain2, water};

C5 = new Gigogne("C5", "CARCEL", new int[]{1});
C5.meshx = new float[]{0.0f, lame};
C5.meshy = C5.meshx;
C5.radius = new float[]{0.0f, 5.76770008e-01f};
C5.media = new Composition[]{fuelGrain2, water};

assmbObj = new Gigogne("assmbObj", "CAR2D", new int[]{5,5});
assmbObj.bihet.setGeometry("SPHE");
assmbObj.bihet.setMixture(fuelGrain1, fuel, new float[]{0.4f, 0.0f},
                          new Composition[][]{{clad1, water, clad1}});
assmbObj.bihet.setMixture(fuelGrain2, fuel, new float[]{0.2f, 0.1f},
                          new Composition[][]{{water, fuel, water},{fuel, clad1, water}});
assmbObj.bihet.setRadius(new float[]{0.0f, 0.1f, 0.2f, 0.3f});
assmbObj.bihet.setRadius(new float[]{0.0f, 0.2f, 0.4f, 0.5f});
assmbObj.bc = new String[][] {{"X-","DIAG"},{"X+","REFL"},{"Y-","SYME"},
                              {"Y+","DIAG"}};
assmbObj.subgeo = new Gigogne[]{ C1, C3, C2, C3, C4,
                                     C3, C3, C3, C4,
                                         C2, C3, C4,
                                             C3, C4,
                                                 C5 };
```

```
assmbObj.merge = new int[]{1,  2,  3,  4,  5,
                               6,  7,  8,  9,
                                  10, 11,  9,
                                      12,  9,
                                          13 };

myScheme = new SimpleCell("myScheme");
myScheme.setGigogne(assmbObj);
myScheme.setMacrolib(myMacro);
myScheme.setType("ROT+");
myScheme.setCylinder("ASKE");
myScheme.setOption("B");
myScheme.setLeakType(new String[]{"B0","SIGS"});
myScheme.run();

source("assertS.bsh");
assertS(myScheme.operatorOut.lcmObj, "B2  B1HOM", 1, -1.970198e-02f);
```

2.4 A multi-parameter compo calculation with Microlib

```
import jargon.*;
myBurnupList = new float[]{9.375f, 18.75f, 37.5f, 75.0f, 500.0f};
myPower = 3.016e17f ; /* flux normalization factor in Mev/(s*cm) */
boronCont = 600.0e-6f ;
//
//---------------------------- new Isotope -----------------------------
// Set the isotopes
U235 =new Isotope("U235", "U235", "U235_4", "APLIB2", "CEA93V4", "U235SS_4");
U238 =new Isotope("U238", "U238", "U238_4", "APLIB2", "CEA93V4", "U238SS_3");
O16  =new Isotope("O16", "O16", "O16_6", "APLIB2", "CEA93V4");
H2O  =new Isotope("H2O", "H2O", "H2O_3_P5", "APLIB2", "CEA93V4");
AL27 =new Isotope("AL27", "AL27", "AL27_4", "APLIB2", "CEA93V4");
PU239=new Isotope("PU239", "PU239", "PU239_4", "APLIB2", "CEA93V4", "PU239SS_4");
PU240=new Isotope("PU240", "PU240", "PU240_4", "APLIB2", "CEA93V4", "PU240SS_4");
PU241=new Isotope("PU241", "PU241", "PU241_4", "APLIB2", "CEA93V4");
PU242=new Isotope("PU242", "PU242", "PU242_3", "APLIB2", "CEA93V4");
AM241=new Isotope("AM241", "AM241", "AM241_1", "APLIB2", "CEA93V4");
AM242M=new Isotope("AM242M", "AM242M", "AM242M_3", "APLIB2", "CEA93V4");
XE135PF=new Isotope("XE135PF", "XE135PF", "XE135PF_1", "APLIB2", "CEA93V4");
//
// Set self-shielding regions
U235.inrs = 1; U235.setIrset(0.0f, 38);
U238.inrs = 1; U238.setIrset(0.0f, 38);
PU239.inrs = 1; PU239.setIrset(0.0f, 38);
PU240.inrs = 1; PU240.setIrset(0.0f, 38);
//
// ---------------------------------------------------------------------
// definition of the depletion isotopic chain
chain_V4 = null;
try {
  int c;
  CharArrayWriter out = new CharArrayWriter();
  File fin = new File("CompoShi.chain");
  FileReader in = new FileReader(fin);
  while ((c = in.read()) != -1) out.write(c);
  chain_V4 = out.toString();
} catch(Exception e) {
  System.out.println("============== error detected =================");
  System.out.println(e);
  return;
}
//
//--------------------------- COMPOSITION ----------------------------
water = new Composition("water", "medium");
water.setComposition(new Isotope[]{H2O}, new float[]{2.3934e-02f});
water.temperature = 27.0f;
fuel1 = new Composition("fuel1", "medium");
fuel1.setComposition(new Isotope[]{U238, U235, O16, PU239, PU240},
                    new float[]{2.2089e-02f, 8.6623e-04f, 4.5910e-02f, 0.0f, 0.0f});
fuel1.temperature = 306.74f;
fuel2 = new Composition("fuel2", "medium");
fuel2.setComposition(new Isotope[]{U238, U235, O16, PU239, PU240},
                    new float[]{2.2089e-02f, 8.6623e-04f, 4.5910e-02f, 0.0f, 0.0f});
fuel2.temperature = 306.74f;
clad = new Composition("clad", "medium");
clad.setComposition(new Isotope[]{AL27}, new float[]{3.9222e-02f});
clad.temperature = 27.0f;
//
//----------------------------- MICROLIB -----------------------------
myMicrolib = new Microlib("myMicrolib");
myMicrolib.ctra = "APOL"; // set type of transport correction
myMicrolib.chain = chain_V4;
myMicrolib.mixs = new Composition[]{water, fuel1, fuel2, clad};
myMicrolib.exec();
//--------------------------------------------------------------------
//
//------------------------------ GIGOGNE -----------------------------
cote = 1.262082f;
lame = 1.322082f;
C1 = new Gigogne("C1", "CARCEL", new int[]{2});
C1.meshx = new float[]{0.0f, cote};
C1.meshy = C1.meshx;
C1.radius = new float[]{0.0f, 3.0e-01f, 4.1266e-01f};
C1.media = new Composition[]{water, water, water};

C2 = new Gigogne("C2", "CARCEL", new int[]{3});
C2.meshx = new float[]{0.0f, cote};
C2.meshy = C2.meshx;
C2.radius = new float[]{0.0f, 3.25296e-01f, 4.60039e-01f, 5.6343e-01f};
C2.media = new Composition[]{clad, clad, clad, water};

C3 = new Gigogne("C3", "CARCEL", new int[]{2});
C3.meshx = new float[]{0.0f, cote};
C3.meshy = C3.meshx;
C3.radius = new float[]{0.0f, 3.0e-01f, 4.1266e-01f};
C3.media = new Composition[]{fuel1, fuel2, water};

C4 = new Gigogne("C4", "CARCEL", new int[]{2});
C4.meshx = new float[]{0.0f, lame};
C4.meshy = new float[]{0.0f, cote};
C4.radius = C3.radius;
C4.media = new Composition[]{fuel1, fuel2, water};

C5 = new Gigogne("C5", "CARCEL", new int[]{2});
C5.meshx = new float[]{0.0f, lame};
C5.meshy = C5.meshx;
```

```
C5.radius = C3.radius;
C5.media = new Composition[]{fuel1, fuel2, water};

myGeom = new Gigogne("myGeom", "CAR2D", new int[]{5,5});
myGeom.bc = new String[][] {{"X-","DIAG"},{"X+","REFL"},{"Y-","SYME"},
                            {"Y+","DIAG"}};
myGeom.subgeo = new Gigogne[]{ C1, C3, C2, C3, C4,
                                   C3, C3, C3, C4,
                                       C2, C3, C4,
                                           C3, C4,
                                               C5 };
myGeom.merge = new int[]{1,  2,  3,  2,  6,
                             2,  2,  4,  6,
                                 5,  4,  6,
                                     4,  6,
                                         7 };
myGeom.text(myMicrolib);
myGeom.exec(myMicrolib);
//
// ----------------------------------------------------------------------
//
//---------------------------------- SYBIL ------------------------------
// Tracking of the geometry for Eurydice-2
myTrack = new Sybil("myTrack");
myTrack.edit = 1;
myTrack.title = "CompoShi tracking with Sybil";
myTrack.maxr = 40;
myTrack.exec(myGeom);
//----------------------------------------------------------------------
//
//------------------------------ Autop -------------------------------
Autop myShi = new Autop("myShi", "Shiba");
   /*myShi.setSimpleSS("W1", U238, new Composition[]{fuel1});
     myShi.setSimpleSS("W2", U238, new Composition[]{fuel2});
     myShi.setSimpleSS("W1", U235);
     myShi.setSimpleSS("W1", PU239);
     myShi.setSimpleSS("W1", PU240);*/
myShi.exec(myMicrolib, myTrack);
//----------------------------------------------------------------------
//
//--------------------------------- COMPO ------------------------------
// Initialize the compo object
myCompo = new Compo("myCompo");
myCompo.setGlobal("BCON", "VALU", "REAL");
myCompo.setGlobal("FTMP", "TEMP", myShi, fuel1);
myCompo.setGlobal("WTMP", "TEMP", myShi, water);
myCompo.setGlobal("BURN", "IRRA");
myCompo.setGlobal("FLUB", "FLUB");
myCompo.setGlobal("PUIS", "POWR");
myCompo.setGlobal("XE1", "CONC", XE135PF, myShi, fuel1);
myCompo.setGlobal("XE2", "CONC", XE135PF, myShi, fuel2);
myCompo.setLocal("burn", "IRRA");
myCompo.setLocal("flub", "FLUB");
myCompo.setLocal("mass", "MASL");
myCompo.setLocal("xe", "CONC", XE135PF);
myCompo.setLocal("mtmp", "TEMP");
myCompo.comment = "'First line of comment'\n'Second line of comment'\n";
myCompo.exec();
```

```
//----------------------------------------------------------------------
//
//--------------------------------- ASM ---------------------------------
myAsm = new Asm("myAsm", "PIJ");
myAsm.ecco = true;
myAsm.exec(myShi, myTrack);
//----------------------------------------------------------------------
//
//--------------------------------- FLUX --------------------------------
myFlux = new Flux("myFlux", "B");
myFlux.leak = new String[]{"B1", "ECCO"};
myFlux.exec(myAsm, myShi, myTrack);
System.out.println("The value of K-EFFECTIVE is " + myFlux.getKeff());
//------------------------------------------------------------------------
//
// Compute the normalization factor
//
myVolume = ((cote*7.0f)+(lame*2.0f))*((cote*7.0f)+(lame*2.0f));
normFct1 = (myPower * 1.60207e-13f) / myVolume;
normFct2 = normFct1 / (2.651005f * 1.00115f);
System.out.println("volume_assemblage= " + this.myVolume + " cm**3. in-fuel power= "
+ normFct2 + " MW/tonne");
System.out.println("normalization power= " + normFct1 + " W/CC");
//
// Burnup loop #####################################################
//
evobeg = 0.0f;
//--------------------------------- EVO ----------------------------------
myEvo = new Evo("myEvo");
myEvo.solution = "RUNG";
myEvo.eps2 = 100.0f;
// ---------------------------------------------------------------------
for (istep=0; istep<myBurnupList.length; istep++) {
  evoend = myBurnupList[istep] / normFct2;
  System.out.println("Burnup step " + (istep+1) + " between " + evobeg + " and "
  + evoend + " day:");
  if (istep == 0) {
    // execution of the burnup operator (first step)
    myEvo.edit = 2;
    myEvo.expm = 1.0f;
    myEvo.satureInit = true;
    myEvo.setDepl(evobeg, evoend, "DAY", "W/CC", normFct1);
    myEvo.exec(myShi, myFlux, myTrack);
  } else {
    // execution of the burnup operator (subsequent steps)
    myEvo.edit = 1;
    myEvo.expm = 1.0e15f;
    myEvo.satureInit = false;
    myEvo.extr = true;
    myEvo.setDepl(evobeg, evoend, "DAY", "W/CC", normFct1);
    myEvo.exec(myEvo, myShi, myFlux, myTrack);
  }
  System.out.println("Nominal flux at step " + (istep+1) + " and at " + evoend + " DAY:");
  //
  // Self-Shielding calculation
  System.out.println("self-shielding at " + evoend + " DAY:");
  //-------------------------------- Autop --------------------------------
  myShi.passes = 1;
  myShi.exec(myShi, myMicrolib, myTrack);
  //----------------------------------------------------------------------
  //
  //-------------------------------- ASM --------------------------------
  myAsm = new Asm("myAsm", "PIJ");
  myAsm.ecco = true;
  myAsm.exec(myShi, myTrack);
  //----------------------------------------------------------------------
  //
  //-------------------------------- FLUX --------------------------------
  myFlux.exec(myFlux, myAsm, myShi, myTrack);
  System.out.println("The value of K-EFFECTIVE is " + myFlux.getKeff());
  //----------------------------------------------------------------------
  float step2 = myBurnupList[istep];
  //
  //------------------------------ EDITION --------------------------------
  Edition myEdit = new Edition("myEdit");
  myEdit.options = new String[]{"POW"};
  myEdit.cond = new int[]{74, 99};
  myEdit.merge = "CELL";
  myEdit.micr2 = new Isotope[]{U235,U238,PU239,PU240,PU241,PU242,
                                     AM241,AM242M,XE135PF};
  myEdit.save = "EDITCDAT   1";
  myEdit.sph = new Bivac("Sph_obj", "DUAL", new int[]{2,2});
  myEdit.exec(myFlux, myShi, myTrack, myGeom);
  //-----------------------------------------------------------------------
  //
  // Normalization to the reactor power
  myEvo.edit = 2;
  myEvo.unsetDepl();
  myEvo.setSave(evoend, "DAY", "W/CC", normFct1);
  myEvo.exec(myEvo, myShi, myFlux, myTrack);
  //
  //-------------------------------- COMPO --------------------------------
  // Compo object construction
  myCompo.edit = 3;
  myCompo.setSet(evoend, "DAY");
  myCompo.setParam("BCON", boronCont);
  myCompo.exec(myCompo, myEdit, myEvo, myShi);
  //-----------------------------------------------------------------------
  evobeg = evoend;
}
//
// Export the compo
myCompo.lcmObj.expor();
System.out.println("CompoShi testcase completed");
```

Note that the depletion chain in file `CompoShi.chain` is defined as

```
 DEPL LIB: APLIB2 FIL: CEA93V4 CHAIN
 U234     FROM N2N      1.0000E+00 U235
 U235     FROM NG       1.0000E+00 U234
 U236     FROM NG       1.0000E+00 U235
 U238
 NP237    FROM NG       1.0000E+00 U236
 PU238    FROM NG       1.0000E+00 NP237    DECAY    1.0000E+00 CM242
 PU239    FROM NG       1.0000E+00 PU238
               DECAY    1.0000E+00 CM243    NG       1.0000E+00 U238
 PU240    FROM NG       1.0000E+00 PU239    DECAY    1.0000E+00 CM244
 PU241    FROM NG       1.0000E+00 PU240
 PU242    FROM NG       1.0000E+00 PU241    NG       1.4160E-01 AM241
 AM241    FROM DECAY    1.0000E+00 PU241
 AM242M   FROM NG       1.1500E-01 AM241
 AM243    FROM NG       1.0000E+00 PU242
 CM242    FROM NG       7.4340E-01 AM241
 CM243    FROM NG       1.0000E+00 CM242
 CM244    FROM NG       1.0000E+00 CM243    NG       1.0000E+00 AM243

 I135PF
 XE135PF  FROM DECAY    1.0000E+00 I135PF
 ND143PF
 ND144PF  FROM NG       1.0000E+00 ND143PF
 ND145PF  FROM NG       1.0000E+00 ND144PF
 ND146PF  FROM NG       1.0000E+00 ND145PF
 ND147PF  FROM NG       1.0000E+00 ND146PF
 ND148PF  FROM NG       1.0000E+00 ND147PF
 PM147PF  FROM DECAY    1.0000E+00 ND147PF
 PM148PF  FROM NG       5.3000E-01 PM147PF
 PM148MPF FROM NG       4.7000E-01 PM147PF
 PM149PF  FROM NG       1.0000E+00 PM148PF  NG       1.0000E+00 PM148MPF
 SM149PF  FROM DECAY    1.0000E+00 PM149PF
 SM150PF  FROM NG       1.0000E+00 SM149PF
 SM151PF  FROM NG       1.0000E+00 SM150PF
 SM152PF  FROM NG       1.0000E+00 SM151PF
 EU153PF  FROM NG       1.0000E+00 SM152PF
 EU154PF  FROM NG       1.0000E+00 EU153PF
 EU155PF  FROM NG       1.0000E+00 EU154PF
 MO95PF TC99PF RH103PF RH105PF
 AG109PF XE131PF CS133PF
 PSU5U PSU8U PSP9U PSP0U PSP1U PSP2U
 ENDCHAIN
```

2.5 The first UO_2 Rowlands benchmark.

This data file corresponds to the first UO_2 Rowlands benchmark solved with the following options:

- Both self-shielding and flux calculations are performed using the long characteristics method of Igor Suslov.[7,8]
- Distributed self-shielding effects are taken into account only for ^{238}U, using six layers in the fuel pin. The Ribon extended method[9] is used for ^{235}U and ^{238}U and a classical subgroup approach (similar to the Helios or Wims-7 approach) is used for Zirconium.
- Cross section data is recovered from an isotopic multigroup library in Draglib format.

```
// UO2 Rowlands benchmark 1 with MCCG
import jargon.*;
//
//---------------------------- new Isotope -----------------------------
// Set the isotopes
U235 = new Isotope("U235", "U235", "U235", "DRAGON", "DLIB_J2");
U238 = new Isotope("U238", "U238", "U238", "DRAGON", "DLIB_J2");
O16  = new Isotope("O16", "O16", "O16", "DRAGON", "DLIB_J2");
H1   = new Isotope("H1", "H1", "H1_H2O", "DRAGON", "DLIB_J2");
ZR   = new Isotope("ZR", "ZR", "Zr0", "DRAGON", "DLIB_J2");
//
// Set non-depleting isotopes and self-shielding regions
```

```
O16.noev = true;
H1.noev = true;
U235.inrs = 1; U235.setRibox(1);
U238.inrs = 1; U238.setRibox(1);
ZR.inrs = 2;   ZR.setRibox();   ZR.noev = true;
//
//--------------------------- COMPOSITION ----------------------------
fuel1 = new Composition("fuel1", "medium");
fuel1.setComposition(new Isotope[]{U238, U235, O16},
                     new float[]{2.2604e-02f, 7.0803e-04f, 4.6624e-02f});
fuel1.temperature = 19.84f;
fuel2 = fuel1.clone("fuel2");
fuel3 = fuel1.clone("fuel3");
fuel4 = fuel1.clone("fuel4");
fuel5 = fuel1.clone("fuel5");
fuel6 = fuel1.clone("fuel6");
clad = new Composition("clad", "medium");
clad.setComposition(new Isotope[]{ZR}, new float[]{4.3241e-02f});
clad.temperature = 19.84f;
water1 = new Composition("water1", "medium");
water1.setComposition(new Isotope[]{H1, O16}, new float[]{6.6988e-2f,
       3.3494e-2f});
water1.temperature = 19.84f;
water2 = new Composition("water2", "medium");
water2.setComposition(new Isotope[]{H1, O16}, new float[]{6.6988e-2f,
       3.3494e-2f});
water2.temperature = 19.84f;
//
//----------------------------- MICROLIB -----------------------------
myMicrolib = new Microlib("myMicrolib", "PTSL");
myMicrolib.ctra = "APOL"; // set type of transport correction
myMicrolib.mixs = new Composition[]{fuel1, fuel2, fuel3, fuel4, fuel5,
                  fuel6, clad, water1, water2};
myMicrolib.edit = 5;
myMicrolib.text();
myMicrolib.exec();
//--------------------------------------------------------------------

cell = new Gigogne("cell", "CARCEL", new int[]{8});
cell.radius = new float[]{0.0f, 0.2529822f, 0.334664f, 0.3577709f,
              0.3794733f, 0.3898718f, 0.40f, 0.45f, 0.5748331f };
cell.meshx = new float[]{0.0f, 1.2f};
cell.meshy = cell.meshx;
cell.media = new Composition[]{fuel1, fuel2, fuel3, fuel4, fuel5, fuel6,
             clad, water1, water2};
cell.bc = new String[][] {{"X-","REFL"},{"X+","REFL"},{"Y-","REFL"},
                          {"Y+","REFL"},};
cell.text(myMicrolib);
cell.exec(myMicrolib);

Mccg myTrack = new Mccg("myTrack");
myTrack.edit = 1;
myTrack.title = "UO2 Rowlands benchmark 1 with MCCG";
myTrack.maxr = 40;
myTrack.trak = "TISO";
myTrack.nangl = 12;
myTrack.dens = 12.0f;
myTrack.symm = -1; // remove automatic symmetry detection
myTrack.moc = "CACB";
myTrack.innerMax = 100;
myTrack.innerEps = 1.0e-5f;
myTrack.krylov = 10;
myTrack.exec(cell);

Autop myShi = new Autop("myShi", "Ribox");
myShi.solution = "ARM";
myShi.setSimpleSS("W1", U238, new Composition[]{fuel1});
myShi.setSimpleSS("W2", U238, new Composition[]{fuel2});
myShi.setSimpleSS("W3", U238, new Composition[]{fuel3});
myShi.setSimpleSS("W4", U238, new Composition[]{fuel4});
myShi.setSimpleSS("W5", U238, new Composition[]{fuel5});
myShi.setSimpleSS("W6", U238, new Composition[]{fuel6});
myShi.setSimpleSS("W1", U235);
myShi.setSimpleSS("W1", ZR);
myShi.text(myMicrolib);
myShi.exec(myMicrolib, myTrack);

myAsm = new Asm("myAsm", "ARM");
myAsm.text();
myAsm.exec(myShi, myTrack);

myFlux = new Flux("myFlux", "K");
myFlux.leak = new String[]{"B0", "SIGS"};
myFlux.exec(myAsm, myShi, myTrack);
System.out.println("The value of K-EFFECTIVE is " + myFlux.getKeff());

source("assertS.bsh");
assertS(myFlux.lcmObj, "K-EFFECTIVE", 1, 1.3924694f);
```

2.6 Calling a parametrized CLE-2000 procedure

In cases where an application software is called from a multi-physics application, it is likely that the multi-physics application will need to call parametrized CLE-2000 procedures (with "`.c2m`" suffix). This approach provides an efficient way of communication between the application software and the multi-physics application. It also permit to develop computational schemes outside the scope (i.e., independently) of the multi-physics application. Parameters are either LCM objects (memory-resident) or files that are managed by the operating system. Multi-physics applications can be programmed in Java using the Jargon framework.

In the following example, a parametrized procedure, `TESTproc.c2m`, take two object parameters and three CLE-2000 input variables. Note that the CLE-2000 variables are always defined after LCM and file objects. The first parameter, `MACRO_ASCII`, is an ASCII file written by the procedure and containing an export of the information pointed by the second parameter `MACRO`. This second parameter is a memory resident LCM object containing a Macrolib. It is accessed in `read-only` mode. The procedure also prints a table-of-content of the root directory of `MACRO`, using the `UTL:` module of the GANLIB. The procedure `TESTproc.c2m` is implemented as

```
REAL KEFF1 KEFF2 ;
INTEGER I123 ;
PARAMETER MACRO_ASCII MACRO ::
    EDIT 1
    ::: SEQ_ASCII MACRO_ASCII ;
    ::: LINKED_LIST MACRO ;
;
:: >>KEFF1<< >>KEFF2<< >>I123<< ;
MODULE UTL: END: ;
*
```

```
UTL: MACRO :: DIR ;
MACRO_ASCII := MACRO ;
ECHO "KEFF1=" KEFF1 ""KEFF2=" KEFF2 "I123=" I123 ;
ECHO "procedure TESTproc completed" ;
END: ;
QUIT "XREF" .
```

More information about the development of CLE-2000 procedures can be found in Ref. 10.

The next Beanshell script is an example of how a multi-physics application can call such a procedure. A LCM object containing a Macrolib is first created by importing its information from an existing ASCII file named _MACRO1. Next, a call to method `my_cle2000.exec()` is performed to execute `TESTproc.c2m`. This example is implemented in ANSI-C in Ref. 5. The corresponding Beanshell script is written

```
import jargon.*;
   System.out.println("Beginning of test");

// create the LCM object containing a Macrolib
   my_lcm = new Jlcm("LCM_IMP", "MACRO1");
   my_lcm.lib();

// construct the lifo stack
   my_lifo = new Lifo();
   my_lifo.push("MACRO_ASCII1", File.class, "ASCII", "my_ascii_file");
   my_lifo.push("MACRO1", my_lcm);
   my_lifo.push("value1", 1.703945f);
   my_lifo.push("value2", 1.562276f);
   my_lifo.push("value3", 12345);
   my_lifo.lib();

// call the parametrized procedure
   my_cle2000 = new Cle2000("TESTproc", my_lifo);
   my_cle2000.exec();

// erase the lifo stack
   while (my_lifo.getMax() > 0) {
     my_node = my_lifo.pop();
     System.out.println("---->" + my_node);
   }
   System.out.println("successful end of execution");
```

2.7 Calling a CLE-2000 procedure with in-out CLE-2000 variables

The CLE-2000 API also offers the possibility to exchange CLE-2000 variables with a procedure. The following CLE-2000 procedure permits to compute the factorial of a number, as proposed in Ref. 10. Here, `n` and `n_fact` are input and output CLE-2000 variable, respectively. The `fact.c2m` procedure is written

```
!
! Example of a recursive procedure.
!
!  input  to  "fact": *n*
! output from "fact": *n_fact*
!
 INTEGER   n n_fact prev_fact ;
  ::  >>n<< ;
 IF n 1 = THEN
   EVALUATE n_fact := 1 ;
 ELSE
```

```
   EVALUATE n := n 1 - ;
   ! Here, "fact" calls itself
   PROCEDURE fact ;
   fact ::   <<n>>  >>prev_fact<< ;
   EVALUATE n_fact := n 1 + prev_fact * ;
 ENDIF ;
  :: <<n_fact>> ;
 QUIT " Recursive procedure *fact* XREF " .
```

This example is implemented in ANSI-C in Ref. 5. The same procedure `fact.c2m` can be called from a Beanshell script, using

```
import jargon.*;
   System.out.println("Beginning of test");

// construct the lifo stack
   my_lifo = new Lifo();
   my_lifo.push("input_val", 5);
   my_lifo.push("output_val", Integer.class);
   my_lifo.lib();

// call the parametrized procedure
   my_cle2000 = new Cle2000("fact", my_lifo);
   my_cle2000.exec();

// erase the lifo stack
   while (my_lifo.getMax() > 0) {
     my_node = my_lifo.pop();
     System.out.println("---->" + my_node);
   }
   System.out.println("successful end of execution");
```

References

[1] 1. G. Marleau, A. Hébert and R. Roy, "New Computational Methods Used in the Lattice Code Dragon," Proc. *Int. Top. Mtg. on Advances in Reactor Physics*, Charleston, USA, March 8–11, 1992.

[2] D. Sekki, A. Hébert and R. Chambon, "A User's Guide for DONJON Version 4," IGE-300, École Polytechnique de Montréal, Institut de Génie Nucléaire, 2009.

[3] A. Hébert, "Coarse-Grain Parallelism Using Remote Method Invocation", Proc. *Int. Conf. on Supercomputing in Nuclear Applications*, Paris, France, September 22–24, 2003.

[4] See `http://www.polymtl.ca/jargon`.

[5] A. Hébert and R. Roy, "The Ganlib Version5 Developer's Guide," IGE-313, École Polytechnique de Montréal, Institut de Génie Nucléaire, 2009.

[6] P. Niemeyer and J. Knudsen, "Learning Java," O'Reilly & Associates, 2002. See also `http://www.beanshell.org/`.

[7] I. R. Suslov, "Solution of Transport Equation in 2– and 3–Dimensional Irregular Geometry by the Method of Characteristics", *Joint Int. Conf. on Mathematical Methods and Supercomputing in Nuclear Applications*, Karlsruhe, Germany, April 19–23, 1993.

[8] I. R. Suslov, "An Algebraic Collapsing Acceleration Method for Acceleration of the Inner (Scattering) Iterations in Long Characteristics Transport Theory", *Int. Conf. on Supercomputing in Nuclear Applications*, Paris, France, September 22–24, 2003.

[9] A. Hébert, "Development of a New Resonance Self-Shielding Methodology Based on Probability Table Information", *Int Mtg. on Nuclear Mathematical and Computational Sciences*, April 6–10, Gatlinburg, Tennessee, 2003.

[10] R. Roy, *The CLE-2000 Tool-Box*, Report IGE–163, Institut de génie nucléaire, École Polytechnique de Montréal, Montréal, Québec (1999).