Dataflow Interchange Format Quick Start Guide

Author: William Plishker
Document version: May 7, 2007

Purpose of this Guide

This guide serves as short set of instructions to show how to use the first release of the *Dataflow Interchange Format* (DIF). It describes the dependencies of the tool and takes you step by step on installing and then exercising a few of the features in this release of DIF.

What this guide is NOT

This is not a way to learn the DIF language. While this guide will show examples Dataflow concepts are not covered. For all of these inquiries we refer you to the latest language reference. This guide will cover complex applications implemented in DIF, but will instead utilize simple applications for illustrative purposes.

Installing DIF

DIF has a few dependencies for complete functionality of DIF:

- A recent version of the Java Development Kit
- Ptolemy II
- GraphViz
- MOCGraph

Install Java Development Kit

Javac will be needed to build applications against the released jar file. The latest JDK can be found at:

http://java.sun.com/javase/downloads/index.jsp

Install Ptolemy II

This release of DIF relies on certain packages present in Ptolemy II. Before you must download and install Ptolemy II. A complete set of instructions can be found at:

http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm

Install GraphViz

To use the visualization features in DIF called DIFDoc, you must have a recent version of GraphViz installed. It can be found at:

http://www.graphviz.org/

MOCGraph

MOCGraph is a graph package tailored to application descriptions based on models of computation. This jar package can be found on our website.

Directory Structure

There is flexibility to the directory structure that can be used to run the framework. For the following steps in this document, the toplevel directory contains the following files and subdirectories

./mapss.jar	The jar containing the DIF package
./mocgraph.jar	The jar containing the MOCGraph package
./ptII/	The Ptolemy II directory
./difs/	Sample DIF files

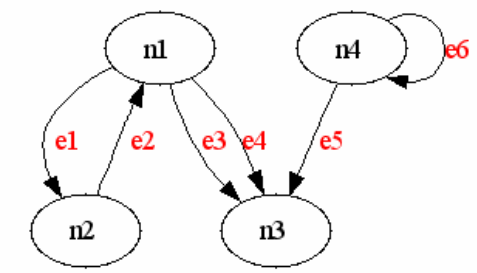
Using the DIF package

Let's start with a simple program which reads in a DIF file and produces some basic information about it. **QuickStartDemo1.java** has the following text:

```
import mapss.dif.DIFHierarchy;
import mapss.dif.language.Reader;
import mapss.dif.language.DIFLanguageException;
import java.io.IOException;

public class QuickStartDemo1 {
    public static void main(String[] args) {
        try {
            Reader reader = new Reader(args[0]);
            reader.compile();
            DIFHierarchy topHierarchy = reader.getTopHierarchy();

            System.out.println(topHierarchy.toString());

        } catch (IOException exp) {
            throw new RuntimeException(exp.getMessage());
        } catch (DIFLanguageException exp) {
            throw new RuntimeException(exp.getMessage());
        }
    }
}
```

This program reads in the first argument from the command line and reads it into the package, creating a *DIFHierarchy* based on it. It then prints some basic information about the graph itself. Consider the following DIF description of an application of in a file called **graph1_1.dif**:

```
dif graph1_1 {
    topology {
        nodes = n1, n2, n3, n4;
        edges = e1 (n1, n2),
                e2 (n2, n1),
                e3 (n1, n3),
                e4 (n1, n3),
                e5 (n4, n3),
                e6 (n4, n4);
    }
}
```



A pictorial representation of graph1_1

Without getting too much into the details of the language structure, it should be apparent how the nodes and edges describe the pictorial representation. To You should be able to compile and run this from a command prompt, ensuring that mapss.jar is in the classpath along with Ptolemy II (ptII) and the location of the QuickStartDemo1:

```
>> javac -cp "mocgraph.jar;ptII;mapss.jar;." QuickStartDemo1.java
>> java  -cp "mocgraph.jar;ptII;mapss.jar;." QuickStartDemo1
difs/graph1_1.dif
name: graph1_1
ports:
sub-hierarchies:
super-hierarchy:
```

The last 4 lines are the output produce by a successful invocation. It indicates **graph1_1.dif** describes a graph named "graph1_1" which has no external ports, no hierarchy. For a slightly more interesting example try **graph1_4.dif**, which has 4 different ports (through the use of the *interface* keyword) and two subgraphs (through the use of the *refinement* keyword):

```
dif graph1_4 {
    topology {
           ...
    }
    interface {
        inputs = p1, p2:n2;
        outputs = p3:n3, p4:n4;
    }
           ...
```

```
        refinement {
            graph2 = n3;
            p1 : e3;
            p2 : e4;
            p3 : e5;
            p4 : p3;
            param1 = param2;
            param2 = param14;
            param5 = param9;
        }

        refinement {
            graph3 = n2;
            p1 : e1;
            p2 : p2;
            p4 : e2;
            param1 = param3;
            param2 = param14;
        }
    }
```

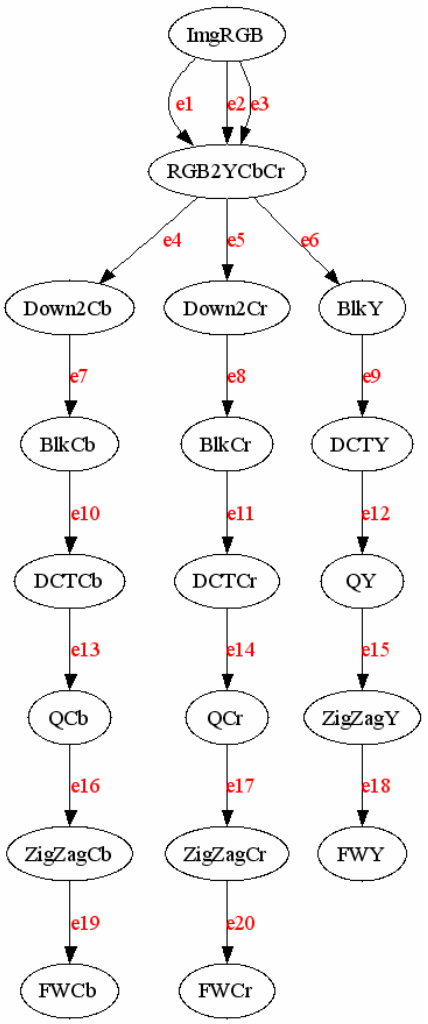
This produces a slightly different output with our example program:

```
>> java -cp "mocgraph.jar;ptII;mapss.jar;." QuickStartDemo1
difs/graph1_4.dif
name: graph1_4
ports: graph1_4.p1 graph1_4.p2 graph1_4.p3 graph1_4.p4
sub-hierarchies: graph2 graph3
super-hierarchy:
```

More DIF can handle real world from a variety of dataflow formats, but perhaps the most popular is synchronous dataflow (SDF). Applications like JPEG (below) and derive useful properties about it such as memory bounds and feasible schedules. DIF accommodates specific dataflow semantics through keywords like *sdf.* For a complete description of the application in specific dataflow models, extra graph information may be required from the such as *production* and *consumption* rates for each of the actors in the case of SDF. This partial description abridged version of a JPEG encoder described in **jpeg.dif**.

```
    sdf jpegEncode {
        topology {
            nodes = ImgRGB,
                    RGB2YCbCr,
                    Down2Cb,
                    ...
            edges = e1 (ImgRGB,RGB2YCbCr),
                    e2 (ImgRGB,RGB2YCbCr),
                    e3 (ImgRGB,RGB2YCbCr),
                    e4 (RGB2YCbCr,Down2Cb),
                    ...
        }
        ...
```

```
        production {
            e1 = 4096;
            e2 = 4096;
            e3 = 4096;
            e4 = 4096;
             ...
        }

        consumption {
            e1 = 4096;
            e2 = 4096;
            e3 = 4096;
            e4 = 4096;
            ...
        }
    }
```



A pictorial representation of jpeg.dif

The edges now have production rates associated with their source actors and consumption rates associated with their sink actors. Based on the semantics SDF, this graph now contains enough information to produce a schedule and bounds on buffer sizes.

With the keyword *sdf* in the DIF description, the DIF package can produce an intermediate representation specific to SDF, allowing the application designers access to features and tools specific to the dataflow model. The following code (**QuickStartDemo2.java**) assumes the input DIF is an SDF application and generates a repetition vector (the relative rates of execution of each actor for a single round of execution of the application). Based on this schedule, it also determines the upper bound on the buffer memory requirements.

```
import mapss.dif.language.Reader;
import mapss.dif.language.DIFLanguageException;
import mapss.dif.csdf.sdf.SDFGraph;

import java.io.IOException;
import java.util.HashMap;
import java.util.Set;
import java.util.Iterator;
import java.util.Collection;

import mocgraph.Node;

public class QuickStartDemo2 {
    public static void main(String[] args) {
        try {
            Reader reader = new Reader(args[0]);
            reader.compile();
            Collection graphs = reader.getGraphs();

            Iterator graphsIterator = graphs.iterator();
            while(graphsIterator.hasNext()){
                SDFGraph graph = (SDFGraph)graphsIterator.next();
                HashMap graphHM = graph.computeRepetitions();
                Set hmSet = graphHM.keySet();
                Iterator hmIter = hmSet.iterator();
                while(hmIter.hasNext()){
                    Node node = (Node)hmIter.next();
                    System.out.println(
                        graph.getRepetitions(node) + "\t" +
                        graph.getName(node));
                }
                System.out.println();
                System.out.println("Buffer memory upper bound = " +
                                   graph.BMUB());
            }
        } catch (IOException exp) {
            throw new RuntimeException(exp.getMessage());
        } catch (DIFLanguageException exp) {
            throw new RuntimeException(exp.getMessage());
        }
    }
}
```

Running this on our implementation of JPEG produces a list of the nodes in the graph preceded by the number of times they must be executed to complete one iteration of the application.

```
>> javac -cp "mocgraph.jar;mapss.jar;ptII;." QuickStartDemo2.java
>> java  -cp "mocgraph.jar;mapss.jar;ptII;." QuickStartDemo2 difs/jpeg.dif
1      BlkCr
64     FWY
1      BlkY
1      BlkCb
1      ImgRGB
64     QY
16     FWCr
1      RGB2YCbCr
16     FWCb
16     ZigZagCb
64     DCTY
16     ZigZagCr
64     ZigZagY
1      Down2Cr
16     DCTCr
16     DCTCb
16     QCr
1      Down2Cb
16     QCb

Buffer memory upper bound = 51200
```

Since the production and consumption rates are balanced on the edges between ImgRGB and RGB2YCbCr, the repetition vector computation found that they execute at the same rate (1 in this case).

DIF Doc

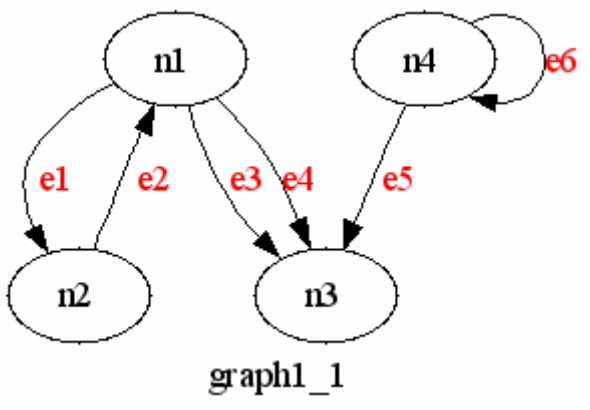
Textual descriptions of graphs are invaluable for understanding and interperating programs described as graphs. DIFDoc can create human readable graphs. As an example, consider the following program called **QuickStartDemo3.java**:

```
import mapss.dif.DIFHierarchy;
import mapss.dif.language.Reader;
import mapss.dif.language.DIFLanguageException;
import java.io.IOException;
import mapss.dif.graph.DIFdoc;

public class QuickStartDemo3 {
    public static void main(String[] args) {
        try {
            Reader reader = new Reader(args[0]);
            reader.compile();
            DIFHierarchy topHierarchy = reader.getTopHierarchy();

            DIFdoc testDoc = new DIFdoc(topHierarchy);
            testDoc.toFile("index");
        } catch (IOException exp) {
            throw new RuntimeException(exp.getMessage());
        } catch (DIFLanguageException exp) {
            throw new RuntimeException(exp.getMessage());
        }
    }
}
```

Compiling and running this at the command line:

```
>> javac -cp "mocgraph.jar;mapss.jar;ptII;." QuickStartDemo3.java
>> java  -cp "mocgraph.jar;mapss.jar;ptII;." QuickStartDemo3
difs/graph1_1.dif
```

produces an *index.html* file in the current directory, which is a textual description of the application with links to graphs generated by GraphViz software. The graphical representation of *graph1_1* linked to by *index.html* is:



Utilizing the MOCGraph package

DIF utilizes a graph package that is tailored to descriptions of applications based on models of computation (MOCs), so members of Ptolemy's can be converted or used directly with the DIF package. The following program (**QuickStartDemo4.java**) converts an application in a *DIFHierarchy* and converts it to a Ptolemy graph.

```
import mapss.dif.DIFHierarchy;
import mapss.dif.language.Reader;
import mapss.dif.language.DIFLanguageException;
import java.io.IOException;
import mocgraph.Graph;

public class QuickStartDemo4 {
    public static void main(String[] args) {
        try {
            Reader reader = new Reader(args[0]);
            reader.compile();
            DIFHierarchy topHierarchy = reader.getTopHierarchy();

            Graph testGraph = topHierarchy.getGraph();
            System.out.println(testGraph.toString());

        } catch (IOException exp) {
            throw new RuntimeException(exp.getMessage());
        } catch (DIFLanguageException exp) {
            throw new RuntimeException(exp.getMessage());
        }
    }
}
```

Which produces the following output for **graph1_1.dif**

```
>> javac -cp "mocgraph.jar;mapss.jar;ptII;." QuickStartDemo4.java
>> java  -cp "mocgraph.jar;mapss.jar;ptII;." QuickStartDemo4
         difs/graph1_1.dif
{mapss.dif.DIFGraph
Node Set:
0: mapss.dif.DIFNodeWeight
1: mapss.dif.DIFNodeWeight
2: mapss.dif.DIFNodeWeight
3: mapss.dif.DIFNodeWeight
Edge Set:
0: (mapss.dif.DIFNodeWeight, mapss.dif.DIFNodeWeight, )
1: (mapss.dif.DIFNodeWeight, mapss.dif.DIFNodeWeight, )
2: (mapss.dif.DIFNodeWeight, mapss.dif.DIFNodeWeight, )
3: (mapss.dif.DIFNodeWeight, mapss.dif.DIFNodeWeight, )
4: (mapss.dif.DIFNodeWeight, mapss.dif.DIFNodeWeight, )
5: (mapss.dif.DIFNodeWeight, mapss.dif.DIFNodeWeight, )
}
```