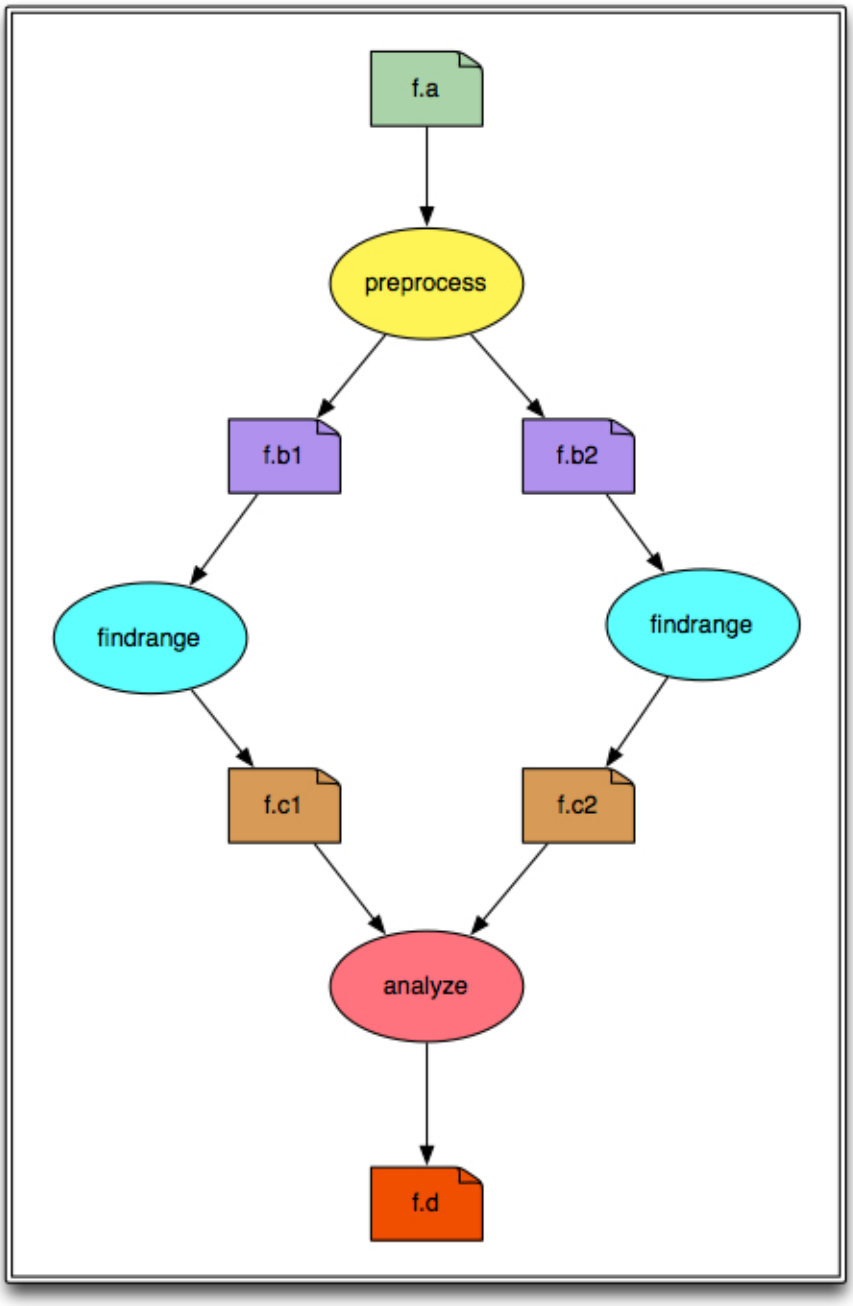
Chapter 1. Generating Abstract Workflows for Pegasus Planner

1. Introduction

Pegasus takes an abstract workflow in XML format called DAX and plans it to run on the grid by refining the workflow. This DAX format has a well-defined schema described at http://pegasus.isi.edu/mapper/docs/schemas/dax-2.1/dax-2.1.xsd and http://pegasus.isi.edu/mapper/docs/schemas/dax-2.1/dax-2.1.html. Below is shown an example workflow we call Diamond followed by the Abstract Workflow representation of it - Diamond Dax.

Figure 1.1. Diamond Dax



```
<adag xmlns="http://pegasus.isi.edu/schema/DAX"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://pegasus.isi.edu/schema/DAX
                          http://pegasus.isi.edu/schema/dax-2.1.xsd"
      version="2.1" count="1" index="0" name="diamond" jobCount="4"
      fileCount="0" childCount="3">

<!-- part 1: list of all referenced files (may be empty) -->

<!-- part 2: definition of all jobs (at least one) -->

  <job id="ID0000001" namespace="diamond" name="preprocess" version="2.0">
    <argument>
        -a preprocess -T60 -i <filename file="f.a"/>
        -o <filename file="f.b1"/> <filename file="f.b2"/>
    </argument>
    <uses file="f.a" link="input" register="true" transfer="true" type="data"/>
    <uses file="f.b1" link="output" register="false" transfer="true" type="data"/>
    <uses file="f.b2" link="output" register="false" transfer="true" type="data"/>
  </job>

  <job id="ID0000002" namespace="diamond" name="findrange" version="2.0">
    <argument>
        -a findrange -T60 -i <filename file="f.b1"/> -o <filename file="f.c1"/>
    </argument>
    <uses file="f.b1" link="input" register="true" transfer="true" type="data"/>
    <uses file="f.c1" link="output" register="true" transfer="true" type="data"/>
  </job>

  <job id="ID0000003" namespace="diamond" name="findrange" version="2.0">
    <argument>
        -a findrange -T60 -i <filename file="f.b2"/> -o <filename file="f.c2"/>
    </argument>
    <uses file="f.b2" link="input" register="true" transfer="true" type="data"/>
    <uses file="f.c2" link="output" register="true" transfer="true" type="data"/>
  </job>

  <job id="ID0000004" namespace="diamond" name="analyze" version="2.0">
    <argument>
         -a analyze -T60 -i <filename file="f.c1"/> <filename file="f.c2"/>
         -o <filename file="f.d"/>
    </argument>
    <uses file="f.c1" link="input" register="true" transfer="true" type="data"/>
    <uses file="f.c2" link="input" register="true" transfer="true" type="data"/>
    <uses file="f.d" link="output" register="true" transfer="true" type="data"/>
  </job>


<!-- part 3: list of control-flow dependencies (may be empty) -->

  <child ref="ID0000002">
    <parent ref="ID0000001"/>
  </child>

  <child ref="ID0000003">
    <parent ref="ID0000001"/>
  </child>

  <child ref="ID0000004">
    <parent ref="ID0000002"/>
    <parent ref="ID0000003"/>
  </child>

</adag>
```

As shown in the example above The DAX consists of 3 sections

- List of all references files in the workflow

The first section which is optional lists all the files being used in the workflow. These files include both input, output and in-out files if they are being produced and consumed by tasks within the workflow.

- Job List

  The second section consists of job or task descriptions which lists each task in the workflow, the logical executable name of the task and the arguments to be given to the task. Also with the arguments are provided a list of input files to the task, and a list out output files produced by the task. You can also attach additional profiles with each job/task

- Control flow dependencies

  The third section consists of dependency links between the tasks. The relationships are defined as child parent relationships. No cyclic dependencies are allowed.

There are three main ways of generating DAX's

- Wings
- DAX Java API
- Direct XML generation

2. DAX API

The Java DAX API provided with the Pegasus distribution allows easy creation of complex and huge workflows. This API is used by several applications to generate their abstract DAX. SCEC, which is Southern California Earthquake Center, uses this API in their CyberShake workflow generator to generate huge DAX containing 10's of thousands of tasks with 100's of thousands of input and output files. The Java API is well documented using Javadocs at http://pegasus.isi.edu/mapper/docs/javadoc/index.html?org/griphyn/vdl/dax/ADAG.html.

The steps involved in creating a DAX using the API are

1. Create a new ADAG object
2. Create a new JOB object
3. Add arguments, files, profiles and other information to the JOB object
4. Add the job object to the ADAG object
5. Repeat step 2-4 as necessary
6. Add Parent child relationships between created jobs to ADAG object.
7. Call the toXML() method on the ADAG object to render the XML DAX file.

An example Java code that generates the diamond dax show above is listed below. This same code can be found in the Pegasus distribution under the examples directory as CreateDax.java

```
import org.griphyn.vdl.classes.LFN;
import org.griphyn.vdl.dax.ADAG;
import org.griphyn.vdl.dax.Filename;
import org.griphyn.vdl.dax.Job;
import org.griphyn.vdl.dax.PseudoText;

import java.io.FileWriter;

public class CreateDAX{


    public static String NAMESPACE ="pegasus";
    public static String VERSION = "2.0";
    public static String PREPROCESS = "preprocess";
    public static String FINDRANGE = "findrange";
```

```
    public static String ANALYZE = "analyze";
    public static String FA="f.a";
    public static String FB1="f.b1";
    public static String FB2="f.b2";
    public static String FC1="f.c1";
    public static String FC2="f.c2";
    public static String FD="f.d";

    public CreateDAX(){
    }

    public void constructDAX(String daxfile){

    try{
        //construct a dax object Giving index of the dax 1 out 1 , with dax label diamond
        ADAG dax = new ADAG(1, 1, "diamond");

        //Create a unqiue id for the first job
        String id1="ID1";

        //create a job with namespace pegasus, name=preprocess, version=2.0 and id=ID1

        Job job=new Job (NAMESPACE,PREPROCESS,VERSION,id1);

        // add the arguments to the job.
        // String types are of type PseudoText and files are of type Filename
        job.addArgument(new PseudoText("-a preprocess "));
        job.addArgument(new PseudoText("-T60 "));
        job.addArgument(new PseudoText("-i "));

        //add the arguments of type File..
        job.addArgument(new Filename(FA));
        job.addArgument(new PseudoText(" -o "));
        job.addArgument(new Filename(FB1));
        job.addArgument(new PseudoText(" "));
        job.addArgument(new Filename(FB2));

        //add the files used by the job explicitly.
        //These may include files not listed on arguments but are still used or produced
        job.addUses(new Filename(FA,LFN.INPUT));
        Filename f=new Filename(FB1,LFN.OUTPUT);
        f.setRegister( false );
        job.addUses(f);
        f=new Filename(FB2,LFN.OUTPUT);
        f.setRegister( false );
        job.addUses(f);

        //Finally add the job to the dax object
        dax.addJob(job);


        //create second  job

        String id2="ID2";
        job=new Job (NAMESPACE,FINDRANGE,VERSION,id2);
        //add the arguments to the job
        job.addArgument(new PseudoText("-a findrange "));
        job.addArgument(new PseudoText("-T60 "));
        job.addArgument(new PseudoText("-i "));
        job.addArgument(new Filename(FB1));
        job.addArgument(new PseudoText(" -o "));
        job.addArgument(new Filename(FC1));
        //add the files used by the job

        job.addUses(new Filename(FB1,LFN.INPUT));
        job.addUses(new Filename(FC1,LFN.OUTPUT));

        //add the job to the dax
        dax.addJob(job);
```

```
        //create third job

        String id3="ID3";
        job=new Job (NAMESPACE,FINDRANGE,VERSION,id3);
        //add the arguments to the job
        job.addArgument(new PseudoText("-a findrange "));
        job.addArgument(new PseudoText("-T60 "));
        job.addArgument(new PseudoText("-i "));
        job.addArgument(new Filename(FB2));
        job.addArgument(new PseudoText(" -o "));
        job.addArgument(new Filename(FC2));
        //add the files used by the job

        job.addUses(new Filename(FB2,LFN.INPUT));
        job.addUses(new Filename(FC2,LFN.OUTPUT));

        //add the job to the dax
        dax.addJob(job);


        //create fourth job

        String id4="ID4";
        job=new Job (NAMESPACE,ANALYZE,VERSION,id4);
        //add the arguments to the job
        job.addArgument(new PseudoText("-a analyze "));
        job.addArgument(new PseudoText("-T60 "));
        job.addArgument(new PseudoText("-i "));
        job.addArgument(new Filename(FC1));
        job.addArgument(new PseudoText(" "));
        job.addArgument(new Filename(FC2));
        job.addArgument(new PseudoText(" -o "));
        job.addArgument(new Filename(FD));

        //add the files used by the job
        job.addUses(new Filename(FC1,LFN.INPUT));
        job.addUses(new Filename(FC2,LFN.INPUT));
        job.addUses(new Filename(FD,LFN.OUTPUT));

        //add the job to the dax
        dax.addJob(job);



        //add the relationships between the jobs (creating a diamond dependency)

        dax.addChild(id2,id1);
        dax.addChild(id3,id1);
        dax.addChild(id4,id2);
        dax.addChild(id4,id3);

        //write DAX to file
        FileWriter daxFw = new FileWriter(daxfile);
        dax.toXML(daxFw, "", null);
        daxFw.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    }

    /**
     * Usage : CreateDAX daxfile
     *
     * @param args the arguments passed
     */
    public static void main(String[] args) {
        CreateDAX daxgen = new CreateDAX();
        if (args.length == 1) {
            daxgen.constructDAX(args[0]);
```

```
        } else {
            System.out.println("Usage: CreateDAX <outputdaxfile>");
        }
    }

}
```

3. DAX generator using Pegasus DAX API

If you are using some other scripting, programming environment other then Java you can directly write out the DAX format using the provided schema using any language, eg. LIGO which is Laser Interferometer Gravitational Wave Observatory application generate their DAX file directly using their own python code. You just need to ensure that the generated XML is well formed and valid with respect to the DAX schema. Plans are to provide a Perl and Python API to generate DAX to make it easier for users to produce their own DAX's.

4. WINGS

WINGS is a knowledge-based framework for creating workflow instances. Given a workflow template and input file descriptions, the system creates valid instantiations of the workflow template. Wings is still under research and development and is available from http://www.isi.edu/ikcap/wings.

5. Workflow Gallery

You can view a bunch of DAX's from various applications here in our Workflow Gallery at http://vtcpc.isi.edu/pegasus/index.php/WorkflowGenerator