Exercises for Lecture 7:

Introduction to Virtual Data Language

In this brief set of exercises we introduce the Virtual Data Language through a few simple examples, run locally on your desktop.

Table of Contents

Part I: "Hello World" Example

Verify that your training login is set up correctly to use the VDS:

```
gk1$ vds-version
1.3.8
gk1$
```

We are using a single, shared virtual data catalog database for the entire class for this exercise, and will use the **namespace** feature of VDL to keep everyone's VDL definitions separate. Here, namespaces serve as a directory-like object, but note that they currently provide no security. Since in these examples we generate VDL for you, we should all be able to safely share the same catalog. Here we search the namespace "air" where we defined some shared transformations that serve as interfaces to the AIR fMRI analysis package (Automated Image Registration), a well as the single UNIX command "echo", which serves for us as a "mock application".

```

gk1$ searchvdc -n air
2005.07.13 15:47:39.877 CDT: [app] searching the database
2005.07.13 15:47:39.880 CDT: [app] wildcard search requested
2005.07.13 15:47:41.439 CDT: [app] found 6 matches total
air::alignlinear
air::align_warp
```

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around

```
air::definecommon_air
air::reslice
air::reslice_warp
air::softmean
gk1$
```

Try doing a similar search of the namespace “unix”.

Now, to begin, copy the exercise directory structure “ex7” to your home directory:

```
gk1$ cp -r /home/vdsdemo/ex7 $HOME
gk1$
```

Change to the “hello world” exercise directory, issue “make clean” to initialize the vdl and configuration files, and examine the **hello.vdl** file for this demo:

```
gk1$ cd $HOME/ex7/hello
gk1$ make clean
gk1$ cat hello.vdl

TR unix::echo( message, output file )
{
  argument = ${message};
  argument stdout = ${file};
}

DV wilde::hello->unix::echo(
        message = "Hello World!",
        file = @{output:"hello.out"}
);
gk1$
```

Then, compile the .vdl file, and generate the “abstract” workflow (.dax file) and the concrete workflow (in this case, a “shell” execution script). The steps to do this are performed (and echoed) by “**make workflow**”, which runs the **vdlc** command to compile the VDL and insert the TR and DV definitions into the virtual data catalog (VDC), and then runs **shplanner** to convert the abstract workflow into an executable shell script. Since this shell script is machine-generated, its somewhat general but rather hard to read. Feel free to inspect it – the main shell script is called **test.sh** and it calls one sub-script file per derivation.

```
gk1$ make workflow
echo "Compiling VDL and building dax:"
Compiling VDL and building dax:
vdlc -u -o hello.dax hello.vdl
2005.07.14 08:02:31.295 CDT: [app] parsing "hello.vdl"
2005.07.14 08:02:33.118 CDT: [app] Trying to add TR unix::echo
2005.07.14 08:02:33.128 CDT: [app] Trying to add DV wilde::hello
```

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around

```
2005.07.14 08:02:33.215 CDT: [app] requesting wilde::hello (1/1)
2005.07.14 08:02:33.246 CDT: [app] saving output to hello.dax
shplanner -o . hello.dax
2005.07.14 08:02:34.287 CDT: [app] will use /opt/vds/etc/dax-1.10.xsd
gk1$ cat hello.dax
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated: 2005-07-14T08:04:06-05:00 -->
<!-- generated by: wilde [??] -->
<adag xmlns="http://www.griphyn.org/chimera/DAX"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.griphyn.org/chimera/DAX
http://www.griphyn.org/chimera/dax-1.10.xsd" version="1.10" count="1"
index="0" name="test" jobCount="1" fileCount="1" childCount="0">
<!-- part 1: list of all referenced files (may be empty) -->
  <filename file="hello.out" link="output"/>
<!-- part 2: definition of all jobs (at least one) -->
  <job id="ID000001" namespace="unix" name="echo" level="1" dv-
namespace="wilde" dv-name="hello">
    <argument>Hello World! </argument>
    <stdout file="hello.out" link="output" varname="file"/>
    <uses file="hello.out" link="output" dontRegister="false"
dontTransfer="false"/>
  </job>
<!-- part 3: list of control-flow dependencies (may be empty) -->
</adag>
gk1$
```

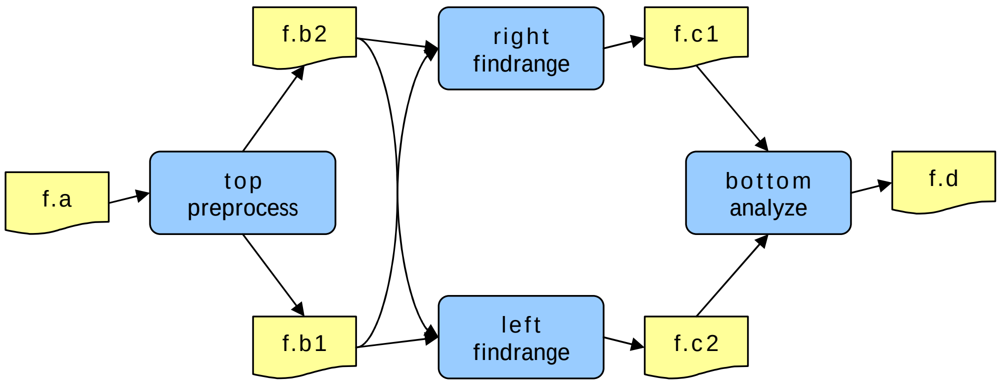
Now, run the workflow:

```
gk1$ ./test.sh
gk1$ ls
gk1$ cat hello.out
```

Exercise: Create a new DV – with a new name – that prints “Hello VIRTUAL World!”.

Part II: Running a simple workflow in VDL

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around



Shown above is the workflow of a simple“diamond” DAG. The “document” symbols represent logical files (f.a, f.b.1, etc), and the bubbles represent jobs. The derivation name is printed bold while the transformation name is printed below. The workflow illustrates the fanning in and fanning out of data, for example, for the parallel execution and reuse of transformations for similar computations. The workflow uses one input file (f.a), which must be registered with in a replica catalog. For these exercises, we use a simple text file for this (located in **$HOME/.vds/rc.data**) that maps between logical and physical filenames. Production workflows typically use the Globus RLS service discussed in Lecture 4. (**The actual file names used here are different!**)

Now, to begin, copy the exercise directory structure “ex7” to your home directory:

Change to the “hello world” exercise directory, issue “make clean” to initialize the vdl and configuration files, and examine the **hello.vdl** file for this demo:

```
gk1$ cd $HOME/ex7/analyze
gk1$ make clean
gk1$ cat analyze.vdl.vdl
gk1$
```

Then, compile the .vdl file, and generate the “abstract” workflow (.dax file) and the concrete workflow (in this case, a “shell” execution script). The steps to do this are performed (and echoed) by “**make workflow**”, which runs the **vdlc** command to compile the VDL and insert the TR and DV definitions into the virtual data catalog (VDC), and then runs **shplanner** to convert the abstract workflow into an executable shell script. Since this shell script is machine-generated, its somewhat general but rather hard to read. Feel free to inspect it – the main shell script is called **test.sh** and it calls one sub-script file per derivation.

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around

```
gk1$ make workflow
gk1$
```

Now, run the workflow:

```
gk1$ ./test.sh
gk1$ ls
gk1$ more *.out
```

Part III – Running a real workflow – Functional MRI Analysis – in the local "shell planner" environment

Initialize VDS Environment (.wfrc, .vdsrc, tc.data, rc.data)

```
gk1$ cd $HOME
gk1$ /home/vdsdemo/wf2/bin/vds-init
gk1$
```

Copy generator and makefile

```
gk1$ cp -r /home/vdsdemo/wf2/vdl $HOME
gk1$
```

Run a "generator" program to expand the VDL for the input dataset:

```
gk1$ cd $HOME/vdl
gk1$ make vdl
./generate > vdsdemo_wf2.vdl
gk1$
```

Examine the VDL – it should have the length indicated below. See if you can discern the graph structure by tracing the output-to-input file dependencies.

```
gk1$ wc -l vdsdemo_wf2.vdl
    205 vdsdemo_wf2.vdl
gk1$ vi vdsdemo_wf2.vdl
```

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around

```
…peruse the file here and then exit vi (or emacs, etc)
gk1$
```

Compile the VDL (which inserts definitions into the VDC and creates an abstract workflow in vdsdemo_wf2.dax. (Since our compiler is still a research tool, its output is terribly voluminous, but this gives you some insight into what its doing).

```
gk1$ cd $HOME/vdl               # still in vdl dir!
gk1$ vdlc -u -l wf2 -o vdsdemo_wf2.dax vdsdemo_wf2.vdl

[training26@gk1 vdl]$ vdlc -u -l wf2 -o vdsdemo_wf2.dax
vdsdemo_wf2.vdl
2005.07.13 16:37:53.917 CDT: [app] parsing "vdsdemo_wf2.vdl"
2005.07.13 16:37:54.892 CDT: [app] Trying to add TR air::alignlinear
2005.07.13 16:37:54.901 CDT: [app] Trying to add TR air::align_warp
2005.07.13 16:37:54.915 CDT: [app] Trying to add TR air::reslice
2005.07.13 16:37:54.926 CDT: [app] Trying to add TR air::reslice_warp
2005.07.13 16:37:54.930 CDT: [app] Trying to add TR
air::definecommon_air
2005.07.13 16:37:54.936 CDT: [app] Trying to add TR air::softmean
2005.07.13 16:37:54.944 CDT: [app] Trying to add TR fsl::slicer
2005.07.13 16:37:54.949 CDT: [app] Trying to add TR fsl::convert
2005.07.13 16:37:54.957 CDT: [app] Trying to add DV
training26::alignlinear_wf2_reference
2005.07.13 16:37:54.971 CDT: [app] Trying to add DV
training26::reslice_wf2_reference
2005.07.13 16:37:55.016 CDT: [app] Trying to add DV
training26::reslice_warp_wf2_reference
2005.07.13 16:37:55.044 CDT: [app] Trying to add DV
training26::align_warp_wf2_reference
2005.07.13 16:37:55.055 CDT: [app] Trying to add DV
training26::alignlinear_wf2_101-3_anonymized
2005.07.13 16:37:55.082 CDT: [app] Trying to add DV
training26::reslice_wf2_101-3_anonymized
2005.07.13 16:37:55.105 CDT: [app] Trying to add DV
training26::reslice_warp_wf2_101-3_anonymized
2005.07.13 16:37:55.117 CDT: [app] Trying to add DV
training26::align_warp_wf2_101-3_anonymized
2005.07.13 16:37:55.146 CDT: [app] Trying to add DV
training26::definecommon_air_wf2
2005.07.13 16:37:55.165 CDT: [app] Trying to add DV
training26::softmean_wf2_sliced-linear
2005.07.13 16:37:55.188 CDT: [app] Trying to add DV
training26::softmean_wf2_sliced-warp
2005.07.13 16:37:55.199 CDT: [app] Trying to add DV
training26::slicer_atlas_-x
2005.07.13 16:37:55.207 CDT: [app] Trying to add DV
training26::slicer_atlas_-y
2005.07.13 16:37:55.218 CDT: [app] Trying to add DV
training26::slicer_atlas_-z
2005.07.13 16:37:55.227 CDT: [app] Trying to add DV
training26::converter_atlas-x.pgm
2005.07.13 16:37:55.235 CDT: [app] Trying to add DV
training26::converter_atlas-y.pgm
```

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around

```
2005.07.13 16:37:55.242 CDT: [app] Trying to add DV
training26::converter_atlas-z.pgm
2005.07.13 16:37:55.291 CDT: [app] requesting
training26::slicer_atlas_-z (1/17)
2005.07.13 16:37:55.545 CDT: [app] requesting
training26::reslice_wf2_101-3_anonymized (2/17)
2005.07.13 16:37:55.546 CDT: [app] Skipping direct request for
already known DV training26::reslice_wf2_101-3_anonymized-
>air::reslice
2005.07.13 16:37:55.547 CDT: [app] requesting
training26::reslice_warp_wf2_reference (3/17)
2005.07.13 16:37:55.547 CDT: [app] Skipping direct request for
already known DV training26::reslice_warp_wf2_reference-
>air::reslice_warp
2005.07.13 16:37:55.548 CDT: [app] requesting
training26::definecommon_air_wf2 (4/17)
2005.07.13 16:37:55.549 CDT: [app] Skipping direct request for
already known DV training26::definecommon_air_wf2-
>air::definecommon_air
2005.07.13 16:37:55.549 CDT: [app] requesting
training26::alignlinear_wf2_101-3_anonymized (5/17)
2005.07.13 16:37:55.553 CDT: [app] Skipping direct request for
already known DV training26::alignlinear_wf2_101-3_anonymized-
>air::alignlinear
2005.07.13 16:37:55.554 CDT: [app] requesting
training26::slicer_atlas_-y (6/17)
2005.07.13 16:37:55.558 CDT: [app] requesting
training26::slicer_atlas_-x (7/17)
2005.07.13 16:37:55.564 CDT: [app] requesting
training26::align_warp_wf2_101-3_anonymized (8/17)
2005.07.13 16:37:55.565 CDT: [app] Skipping direct request for
already known DV training26::align_warp_wf2_101-3_anonymized-
>air::align_warp
2005.07.13 16:37:55.566 CDT: [app] requesting
training26::converter_atlas-z.pgm (9/17)
2005.07.13 16:37:55.571 CDT: [app] requesting
training26::softmean_wf2_sliced-warp (10/17)
2005.07.13 16:37:55.572 CDT: [app] Skipping direct request for
already known DV training26::softmean_wf2_sliced-warp->air::softmean
2005.07.13 16:37:55.572 CDT: [app] requesting
training26::align_warp_wf2_reference (11/17)
2005.07.13 16:37:55.573 CDT: [app] Skipping direct request for
already known DV training26::align_warp_wf2_reference-
>air::align_warp
2005.07.13 16:37:55.574 CDT: [app] requesting
training26::softmean_wf2_sliced-linear (12/17)
2005.07.13 16:37:55.578 CDT: [app] Skipping direct request for
already known DV training26::softmean_wf2_sliced-linear-
>air::softmean
2005.07.13 16:37:55.578 CDT: [app] requesting
training26::alignlinear_wf2_reference (13/17)
2005.07.13 16:37:55.579 CDT: [app] Skipping direct request for
already known DV training26::alignlinear_wf2_reference-
>air::alignlinear
2005.07.13 16:37:55.579 CDT: [app] requesting
training26::converter_atlas-y.pgm (14/17)
```

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around

```
2005.07.13 16:37:55.585 CDT: [app] requesting
training26::converter_atlas-x.pgm (15/17)
2005.07.13 16:37:55.590 CDT: [app] requesting
training26::reslice_wf2_reference (16/17)
2005.07.13 16:37:55.651 CDT: [app] Skipping direct request for
already known DV training26::reslice_wf2_reference->air::reslice
2005.07.13 16:37:55.651 CDT: [app] requesting
training26::reslice_warp_wf2_101-3_anonymized (17/17)
2005.07.13 16:37:55.655 CDT: [app] Skipping direct request for
already known DV training26::reslice_warp_wf2_101-3_anonymized-
>air::reslice_warp
2005.07.13 16:37:55.656 CDT: [app] saving output to vdsdemo_wf2.dax
[training26@gk1 vdl]$
```

Generate a visual rendering of the abstract workflow graph (DAG). Can you spot the dependencies you looked for in the VDL? (Sorry, the resolution of this image is low – you wont be able to read the text… we're still developing our interfaces to the "graphviz" toolkit).

```
gk1$ cd $HOME/vdl
gk1$ make graph
```

The workflow graph can be viewed through your Firefox browser at:
http://172.16.82.207:8080/~username
where username is your UNIX if on gk1 (trainingN).

```
gk1$ shplanner -o vdsdemo_wf2_run_01 vdsdemo_wf2.dax
gk1$
```

Now, execute the actual workflow produced by the shell planner. This will perform several minutes of real fMRI computation (image analysis) on many megabytes of input data (still a very "tiny" example for this field):

```
gk1$ cd vdsdemo_wf2_run_01
gk1$ ./wf2.sh & tail –f wf2.log     # don't forget the "./"  !
2005-07-14T08:47:12 ID000002 started air::alignlinear
2005-07-14T08:47:51 ID000002 finished air::alignlinear
2005-07-14T08:47:51 ID000001 started air::alignlinear
2005-07-14T08:47:57 ID000001 finished air::alignlinear
2005-07-14T08:47:57 ID000003 started air::definecommon_air
2005-07-14T08:47:57 ID000003 finished air::definecommon_air
2005-07-14T08:47:57 ID000004 started air::reslice
```

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around

Terminology

A few terms that we will use here include:

VDL – *Virtual Data Language* – a language that declares the inputs and outputs of executable programs (“Transformations”) and how specific calls to them are chained together into “workflow graphs”

VDLt – a textual version of VDL, loosely inspired by C

VDLx – an XML version of VDL

VDS – *Virtual Data System* – the set of tools used to process VDL statements into a form in which they can be executed on the Grid, or tested locally.

VDC – *Virtual Data Catalog* – a database that holds VDL definitions. The VDC can be a plain file of XML or a relational database. PostgreSQL and MySQL are supported.

Chimera – A project name for the virtual data language toolset

TR – *Transformation* – an application program whose input and out files and parameters are described in VDL

DV – *Derivation* – a call to a transformation, with actual arguments specified (constants and logical files)

LFN – *Logical File Name* – the name of a set of replicas of a given file, each of which may have a unique physical file name on system on which it resides.

Replica catalog: a catalog that maps logical file names on to physical file names

Transformation catalog: a catalog that maps transformation names onto the physical pathname of the transformation at a given gird site or local test machine.

Graph – (informally) in computer science and discrete mathematics, a data structure consisting of linked nodes.

DAG – Directed Acyclic Graph – a graph in which all the arcs (connections) are unidirectional, and which has no loops (cycles). In other words, control can only flow through the graph in one direction, from its start towards its end. Often, in the VDS, when we say “DAG” we mean a file in Condor “DAGman” input format.

DAX – DAg in Xml format. The DAG format produced by the “gendax” tool as it traverses a Virtual Data Catalog to produce a derivation.

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around

Workflow – a graph consisting of tasks, often represented as a DAG.

Abstract workflow – In the VDS: a workflow in which transformations and files are represented as logical names, and in which the location at which to run each transformation is not yet determined.

Concrete workflow – In the VDS: a workflow in which files are represented by physical filenames, and in which sites or hosts have been selected for running each task.

Planner – In the VDS, a tool that converts a workflow graph to a form “more ready” for execution. It makes “plans” for where, how, and when the actions of the workflow should take place.

Shell planner – a planner (“shplan”) that converts a workflow in the form of a VDL “DAX” to a simple, sequential shell script, for execution. It honors all the dependencies in the original VDL.

Formatted: Position: Horizontal: Center, Relative to: Margin, Vertical: 0", Relative to: Paragraph, Wrap Around