Rule-Based Workflow

John Conery
University of Oregon

Research Goals

• This talk: a new method for organizing, executing bioinformatics workflows
• Main goal: a system to support the development and maintenance of workflows
  ° a framework for creating workflows for new projects
  ° support for modifying, extending existing workflows
• An effective system for workflow development
  ° supports an iterative software development methodology
  ° promotes modularity and reusability of workflow components

Outline

- Introduction: workflow systems, workflows in bioinformatics
- Rule-based workflow
  ° software architecture: data-centric pipeline
  ° relational database for workflow products
  ° rule-based workflow enactor
- Support for iterative design
  ° rule structure
  ° stage objects and inheritance

Workflow Management

• The term “workflow” refers to large scale operations involving a combination of activities, or steps
  ° WFMS schedules steps, manages data
  ° common application: business process management
  ° workflow may involve human interaction in addition to computer applications
• Workflow Management Coalition
  ° reference model: http://www.wfmc.org/standards/model.htm
• See also van der Aalst and Hee, Workflow Management, MIT Press (2002)
**Scientific Workflow**

- "e-science" workflow systems coordinate computational science data and applications
- Early systems: problem solving environments
  - e.g. Molecular Biology Workbench
    [http://workbench.sdsc.edu](http://workbench.sdsc.edu)
- Recent research: Grid-enabled workflow
  - applications, data sources reside anywhere on the internet
  - Liu and Franklin, GridDB, *VLDB* ’04 [physics/astronomy]

**Bioinformatics Workflows**

- Bioinformatics is especially well-suited for workflow systems
  - NSF/NLM Workshop on Data Management for Molecular and Cell Biology (LBNL-52767), 2003
- In the last two years alone:
  - BioMake (BDGP)
  - BioPipe (part of Bio::Perl)
  - BioWBI (IBM)
  - HyperThesis (Singapore)
  - Pegasys (UBC)
  - Taverna (BioGrid/EMBL)

**Bioinformatics Workflows (cont’d)**

- A project may use BLAST, CLUSTALW, MrBayes, variety of other tools
- For a small project (e.g. study gene family):
  - run programs “by hand”, e.g. use web browser to connect to BLAST server
  - cut-and-paste results from one app into files to use as inputs for other apps
- For a bigger project (e.g. full-genome analysis):
  - automate as much as possible

**Why Automate Workflow?**

- Laziness
- Nerdiness
- Reproducible results
  - it’s far too easy to make a mistake
  - forget a step, use the wrong parameter, ...
- Formal record of the experimental protocol
  - workflow specification language is important
  - want simple, declarative description, not ad hoc Perl script
Workflow Models

• The majority of workflow systems use a “flowchart” model
  ◦ a graph represents steps and interconnections
  ◦ workflow enactor schedules jobs according to the graph
  ◦ complex workflows involve iteration, conditionals (XOR nodes, joins, merges, ...)

• Analyzing, modeling workflows:
  ◦ BioWBI: 5 simple patterns plus algebra
  ◦ Process logics

Alternative Models

• van der Aalst, et al (YAWL):
  ◦ use Petri net to model workflow
  ◦ steps (transitions) require resources (tokens)
  ◦ formalism helps analyze workflow states

  • For complex workflows, starting an arbitrary process can be difficult
    ◦ what resources does it need?

  • Iterative development and maintenance requires ability to start/restart at any point

  • Data-centric workflow meets this requirement
    ◦ Liu et al, GridDB

Iterative Design and Workflows

• Iterative design is effective
  ◦ implement, test first step of the workflow
  ◦ add next step only when the first is working correctly

• Iterative design is inevitable
  ◦ new sources of data, or new data formats
  ◦ new releases of software
  ◦ results suggest further analyses

Data-Centric Pipeline

• Our approach is based on a new software architecture for bioinformatics workflows

• In a data-centric pipeline:
  ◦ work products are stored in a database (e.g. MySQL)
  ◦ workflow steps read/write streams of records
  ◦ one-to-one relationship between workflow steps and database tables
    ◦ every step produces a table
    ◦ no table is updated by a later step
Relational Databases

- Why a database? Why RDB?
  - regular structure for application I/O (streams of records)
  - separation of concerns: RDB deals with formats and representations, applications deal with algorithms and analysis
  - RDB can implement many functions directly, e.g. “rbh”

- Maybe most important:
  - queries help explore/analyze/understand data
  - an essential part of workflow construction

PIP

- PIP (pipeline interface program) is a workflow management system
  - works within the data-centric pipeline framework
  - written in Perl, connects to MySQL (RDBMS)
  - workflow steps defined by rules
  - steps are scheduled automatically when tables they depend on are updated

PIP Rules

- Rules for steps are similar to rules in a Makefile:
  
  \[ x: \ a \ b \] commands
  
  - x is the name of a table in the database (aka “target”)
  - a and b are also tables, sources of data for the application(s) that builds x
  - in simple rules the commands that build the table are shell commands
  - later: how to use object-oriented programming concepts to write rule patterns, reuse commands from other rules

Checking Dependences

- When a rule is invoked, PIP first checks the timestamps on the dependence tables
  
  \[ x: \ a \ b \] commands
  
  - to see if x needs to be updated, recursively invoke the rules for a and b
  - if either a or b is newer than x (or x does not exist) the commands are executed
Example Project: tRNAMart

• PIP is used to manage the applications that build the database behind the tRNA DataMart

Main goal: produce reports in formats read by other applications (FASTA, CSV, PDF, ...)

The tables in the database are shown at right

There is one PIP rule for each table

Workflow:
° download genome files from NCBI
° run trnascan-SE
° local apps (align, codon usage, etc)

If new species at NCBI: restart the source step

If we decide to use different criteria for tRNA: restart the trna step
tRNAMart (cont’d)

- From the datamart workflow: a rule to build a list of words found in the taxonomy (phylum, order, ...):

```sql
CREATE TABLE IF NOT EXISTS treewords

mysql $DB < mysql/treewords.sql
```

```sql
mysql $DB -N -s -e "SELECT lineage FROM taxonomy ORDER BY lineage" | taxscan > treewords.txt
```

```sql
mysqlimport $DB -L treewords.txt
```

```bash
rm -f treewords.txt
```

(The body of this rule is a set of Unix commands that build a treewords table using data from a taxonomy table)

Running PIP

- Workflow rules are stored in a “pipfile”
- Invoke PIP from the shell

```bash
% pip x
```

- rebuild x (and any out of date dependences)

```bash
% pip -n x
```

- check dependences, but don’t execute rule bodies

```bash
% pip -r x
```

- rebuild x (and only x) whether it’s out of date or not

Checking Dependences (cont’d)

- A rule that has no dependences is always enabled

```bash
x:
  commands
```

- A special symbol marks tables at the start of the workflow (otherwise they would always be updated); run with `pip -r`

```bash
x:
  @ commands
```

- A “phony” rule is one used for control flow (it does not have a corresponding table)

```bash
.phony: all
all:   treewords aligned codons GC
```

Stage Objects

- The commands in the excluded example illustrate a common pattern
- This pattern (“prep, step, grep”) is implemented by a `Stage` object
- In the body of a rule, `Stage("q","c")` means:
  - instantiate a new object of the Stage class
  - invoke the exec method of the class (execute the stage), which in turn invokes `prep`, `step`, and `grep`
  - anything printed to stdout is captured and inserted into the table named by the rule
Stage Object Example

- The treewords rule from a previous example:

```plaintext
treewords: taxonomy
  mysql $DB -e "DROP TABLE IF EXISTS treewords"
  mysql $DB < mysql/treewords.sql
  mysql $DB -N -s -e "SELECT lineage FROM taxonomy ORDER BY lineage" |
  taxscan > treewords.txt
  mysqlimport $DB -L treewords.txt
  rm -f treewords.txt
```

- The same rule with a Stage object:

```plaintext
treewords: taxonomy
  Stage(
    "SELECT lineage FROM taxonomy ORDER BY lineage",
    "taxscan"
  )
```

prep

- The treewords rule from a previous example:

```plaintext
treewords: taxonomy
  mysql $DB -e "DROP TABLE IF EXISTS treewords"
  mysql $DB < mysql/treewords.sql
  mysql $DB -N -s -e "SELECT lineage FROM taxonomy ORDER BY lineage" |
  taxscan > treewords.txt
  mysqlimport $DB -L treewords.txt
  rm -f treewords.txt
```

- The same rule with a Stage object:

```plaintext
treewords: taxonomy
  Stage(
    "SELECT lineage FROM taxonomy ORDER BY lineage",
    "taxscan $DB"
  )
```

tool

- The treewords rule from a previous example:

```plaintext
treewords: taxonomy
  mysql $DB -e "DROP TABLE IF EXISTS treewords"
  mysql $DB < mysql/treewords.sql
  mysql $DB -N -s -e "SELECT lineage FROM taxonomy ORDER BY lineage" |
  taxscan > treewords.txt
  mysqlimport $DB -L treewords.txt
  rm -f treewords.txt
```

- The same rule with a Stage object:

```plaintext
treewords: taxonomy
  Stage(
    "SELECT lineage FROM taxonomy ORDER BY lineage",
    "taxscan $DB"
  )
```
Derived Classes

- Programmers can implement their own classes
  - usual Perl package structure
  - base class Stage is defined in PIP
- Provide new definitions for exec or any of the three phases (prep, step, or grep)
- Import the new class with a PIP directive, e.g.

```perl
.require download.pm
download:
  source
  Download(
    "SELECT url, ftpdir, ftpat FROM source",
    "download -v"
  )
```

Workflow Design: Defining a New Step

1. Write a query that selects information needed for the step
   - run MySQL interactively
     ```
     % mysql tRNA
     mysql> SELECT * FROM taxonomy LIMIT 10;
     ```
   - or from the shell
     ```
     % mysql tRNA -N -s -e "SELECT...."
     ```

Defining a New Step (cont’d)

2. Test the application
   - pipe the query output to the application
     ```
     % mysql tRNA -N -s -e "SELECT...." | taxscan
     ```

Defining a New Step (cont’d)

3. Design the table that will hold results from the step
   - several programs (e.g. CocoaMySQL for Mac OS/X) have a GUI front-end to MySQL for defining new tables

4. Collect parts 1-3 into a rule for the Pipfile
The Evolution of a Project

- A scenario for adding a new step (x) to a project:
  - implement, test x (steps 1-4 from previous slides)
  - use pip -r x as necessary when testing
  - if implementation of x requires modifications to a previous step y, update and test with pip -r y
  - examine the effects of changes to y on the rest of the workflow: pip -n all
  - bring the workflow up to date: pip all

Advantages of Rule-Based Workflow

- A rule-based approach supports reusability
  - copy, revise rules from another project
  - PIP’s object-oriented rules allow inheritance
- The data-centric, rule-based system supports iterative development
  - process-centric systems lack explicit workflow state
  - difficult to analyze
  - difficult to restart (e.g. when testing)

Summary

- New software architecture for workflows
  - data-centric pipeline
  - developed for bioinformatics, but should work in other domains
- Rule-based workflow enactor (PIP)
- Intended to be used for design, implementation, testing of workflows
  - rules are inherently modular, easily imported
  - object orientation also promotes reuse

For More Information

- New paper:
- PIP web site: http://teleost.cs.uoregon.edu/pip
  - pip application (requires Perl, MySQL)
  - manual and tutorial