Scientific Experiments as Workflows and Scripts

Vanessa Braganholo
The experiment life cycle

Source: Marta Mattoso, IJBIMP, 2010
Agenda

• Abstract Representation of Scientific Experiments
• Workflows
• Scripts
• Black Boxes X White Boxes
• Workflow Management Systems
• Provenance Management Systems for Scripts
Composition: Conceiving Scientific Experiments

- Scientists usually design an experiment using a high abstraction level representation that is later mapped into a workflow or script
Phylogeny Analysis Experiment (Abstract Workflow)

1. MSA Construction
2. Evolutionary Model Election
3. Construction of Phylogenetic Tree
4. MSA Conversion
5. MSA Concatenation
6. Selection of Evolutionary Model
7. Construction of Phylogeny Tree

Abstract x Concrete

• The abstract workflow is later mapped into a concrete workflow or script
import vtk

data = vtk.vtkStructuredPointsReader()
data.SetFileName("../.../examples/data/head.120.vtk")

contour = vtk.vtkContourFilter()
contour.SetInput(0, data.GetOutput())
contour.SetValue(0, 67)

mapper = vtk.vtkPolyDataMapper()
mapper.SetInput(contour.GetOutput())
mapper.ScalarVisibilityOff()

actor = vtk.vtkActor()
actor.SetMapper(mapper)

cam = vtk.vtkCamera()
cam.SetViewUp(0, 0, -1)
cam.SetPosition(745, -453, 369)
cam.SetFocalPoint(135, 135, 150)
cam.ComputeViewPlaneNormal()

ren = vtk.vtkRenderer()
ren.AddActor(actor)
ren.SetActiveCamera(cam)
ren.ResetCamera()

renwin = vtk.vtkRenderWindow()
renwin.AddRenderer(ren)

style = vtk.vtkInteractorStyleTrackballCamera()
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renwin)
iren.SetInteractorStyle(style)
iren.Initialize()
iren.Start()
Scientific Workflow

- A scientific workflow is a chain of activities organized in the form of a data flow
Data Flow

• In a data flow, the execution is guided by the data
• As soon as all the input data of an activity is available, it starts executing
Example

- Activities `vtkStructuredPointsReader` and `vtkCamera` do not depend on other activities data, so they can start executing right away.

Script

• Definition is controversial
• One of the most accepted definitions is that a script language is a “programming language that does not require an explicit compilation step”
• In other words, scripts are usually written in Languages that are interpreted instead of compiled
• Examples: Python, R, MatLab, etc.
Script

- Execution follows a **control flow** instead of a data flow
  - Commands explicitly define the execution order
DRY_RUN = ...

def process(number):
    while number >= 10:
        new_number, str_number = 0, str(number)
        for char in str_number:
            new_number += int(char) ** 2
        number = new_number
    return number

def show(number):
    if number not in (1, 7):
        return "unhappy number"
    return "happy number"

n = 2 ** 4000
final = process(n)
if DRY_RUN:
    final = 7
print(show(final))

Source: Pimentel et al., 2016. Fine-grained Provenance Collection over Scripts Through Program Slicing
Running an Experiment

• A workflow or script is just part of an experiment
• In order to prove or refute an hypothesis, it is usually necessary to run the workflow or script several times, varying inputs, parameters and programs
• Each of those runs is called a trial of the experiment
New experiment!

Could you check if the precipitation of Rio de Janeiro remains constant across years?
1st Iteration

- $H↓1$: “The precipitation for each month remains constant across years”

Project  Data: 2013, 2014 [BDMEP]

- experiment.py
- precipitation.py
- p13.dat
- p14.dat

Diagram:
- Composition
- Analysis
- Execution
1| `import` `numpy` `as` `np`
2| `from` `precipitation` `import` `read`, `sum_by_month`
3| `from` `precipitation` `import` `create_bargraph`
4|
5| `months = np.arange(12) + 1`
6|
7| `d13, d14 = read("p13.dat"), read("p14.dat")`
8|
9| `prec13 = sum_by_month(d13, months)`
10| `prec14 = sum_by_month(d14, months)`
11|
12| `create_bargraph("out.png", months, [
13| "2013", "2014"],
14| `prec13, prec14`)`
$ now run -e Tracker experiment.py
Conclusion: “Drought in 2014”
2\textsuperscript{nd} Iteration

- $H↓2$ : “The precipitation for each month remains constant across years \textbf{if there is no drought}”

Project  Data: \textbf{2012, 2013, 2014 [BDMEP]}

- experiment.py
- precipitation.py
- p12.dat
- p13.dat
- p14.dat
```python
import numpy as np
from precipitation import read, sum_by_month
from precipitation import create_bargraph

months = np.arange(12) + 1
d12 = read("p12.dat")
d13, d14 = read("p13.dat"), read("p14.dat")
prec12 = sum_by_month(d12, months)
prec13 = sum_by_month(d13, months)
prec14 = sum_by_month(d14, months)
create_bargraph("out.png", months,
["2012", "2013", "2014"],
prec12, prec13, prec14)
```
$ now run -e Tracker experiment.py
Version Model

Product Space

- Project
  - experiment.py
  - precipitation.py
  - p12.dat
  - p13.dat
  - p14.dat
  - out.png
  - provenance

Version Space

<table>
<thead>
<tr>
<th>Trial</th>
<th>Trial 1</th>
<th>Trial 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Trial History

- Trial 1
- Trial 2

write
read
Conclusion:
“2012 was similar to 2013”
I don’t think it’s enough to compare just these years. Could you add data from 2015?
The same can be done for workflows.
The same can be done for workflows.

Each of these can originate several trials.

Trials in Workflows
History Graph (VisTrails)

Several ways to go from abstract to concrete

• When using scripts, there are several ways to go from abstract to concrete workflows
  – Activities are implemented one after the other in the script (no functions)
  – Activities are mapped into functions (each activity becomes one or more function)
Black Box X White Box

• In Workflow systems, activities are black boxes
  – What goes in and out are known, but what happens inside is not known

• In scripts, activities can be black boxes or white boxes
  – An activity in a script can call an external program, and in this the activity is a black box
  – When the function is implemented in Python, it is a white box
Black Box X White Box

• Black boxes have implications in provenance analysis
Which values influence the result printed by this script? (variable `final`)

Source: Pimentel et al., 2016. Fine-grained Provenance Collection over Scripts Through Program Slicing
DRY_RUN = ...

def process(number):
    while number >= 10:
        new_number, str_number = 0, str(number)
        for char in str_number:
            new_number += int(char) ** 2
        number = new_number
    return number

def show(number):
    if number not in (1, 7):
        return "unhappy number"
    return "happy number"

n = 2 ** 4000
final = process(n)

if DRY_RUN:
    final = 7
print(show(final))

If DRY-RUN is 7, then final depends only on DRY_RUN
If not, then final also depends on n

Source: Pimentel et al., 2016. Fine-grained Provenance Collection over Scripts Through Program Slicing
Implications of Black Boxes

- If `process(number)` were a black box, anything could happen inside it
- It could, for example, read a file that could influence the value returned by the function, so dependencies would be missed
- This is a common case of *implicit provenance*, that is missed by several provenance capturing approaches
Implicit Provenance

Sources:
Implicit Provenance

- OS-Based approaches are able to capture this kind of provenance
- Other approaches need special components to handle it (e.g. PROVMONITOR)
Overview of Existing Systems

• Workflow Management Systems
• Provenance Management Systems for Scripts
Workflow Management Systems

• VisTrails
• Taverna
• Kepler
• Swift
• SciCumulus
• Pegasus
• …
VisTrails

- Visual drag and drop interface for workflow composition
- Captures history of changes in the workflow structure
- Allows comparing results side-by-side
- Focus on visualization
VisTrails
Taverna

- Focus on Bioinformatics
- Several ready-to-use bioinformatics services
- Drag and Drop graphical interface for workflow composition

http://www.taverna.org.uk/
Kepler

- Drag and Drop graphical interface for workflow composition
- Different actors that rules how the workflow executed – Kepler workflows are not DAG

https://kepler-project.org/
Swift, SciCumulus and Pegasus

- Focus on High Performance
- Workflows are specified in XML (no graphical interface) in SciCumulus and Pegasus
- In Swift, workflows are specified as scripts in a specific language

http://swift-lang.org/main/index.php
https://scicumulusc2.wordpress.com/
https://pegasus.isi.edu/
Provenance Management Systems for Scripts

- noWorkflow
  - captures provenance for Python scripts
- RDataTracker
  - captures provenance for R scripts
- Sumatra
  - captures provenance for Python, R and MatLab scripts
Exercise

• Choose one of the systems presented in today’s class and search the Web to find:
  – What is the format in which provenance is stored
  – If they export provenance in the PROV format
Provenance of these slides