

Provenance Challenge 3: VisTrails

University of Utah

David Koop
Tommy Ellkvist
Juliana Freire
Cláudio Silva



Workflow Implementation

The screenshot displays the VisTrails Builder interface for a workflow named 'workflow.vt+'. The interface includes a top toolbar with icons for New, Open, Save, Undo, Redo, Execute, Pipeline, History, Query, Exploration, Select, Pan, and Zoom. On the left, a 'Modules' panel lists various modules under 'Provenance Challenge 3' and 'Control Flow'. The central workspace shows a workflow graph with nodes such as 'String', 'ConcatenateString', 'CreateEmptyLoadDB', 'Group', 'Map', 'GetCSVFiles', 'Module', 'CompactDatabase', 'DetectionsHistogram', 'MplPlot', 'MplFigure', and 'MplFigureCell'. A 'Methods' panel on the right shows a table with columns 'Method' and 'Signature'. The 'Set Methods' panel is also visible.

Modules

Provenance Challenge 3

- CreateEmptyLoadDB
- ReadCSVFile
- DetectionsHistogram
- GetCSVFiles
- IsMatchCSVFileColumnNames
- LoadCSVFileIntoDB
- ReadCSVReadyFile
- IsMatchCSVFileTables
- DatabaseEntry
- CSVFileEntry
- IsMatchTableColumnRanges
- IsMatchTableRowCount
- IsExistsCSVFile
- CompactDatabase
- UpdateComputedColumns
- ComputeColumns
- ReadCSVFileColumnNames
- LoadCSVFileIntoTable
- IsCSVReadyFileExists
- Collection

Control Flow

- ListOfElements
- Fold

Methods

Method	Signature
--------	-----------

Set Methods

Workflow Implementation

The image displays the VisTrails Builder interface for a workflow named 'workflow.vt*'. The main window features a toolbar with icons for New, Open, Save, Undo, Redo, Execute, Pipeline, History, Query, Exploration, Select, Pan, and Zoom. A 'Modules' panel on the left lists various data processing and control flow modules under 'Provenance Challenge 3' and 'Control Flow'. The central workspace shows a workflow diagram with nodes: 'String', 'ConcatenateString', 'CreateEmptyLoadDB', 'Group', and 'Map'. A 'Methods' panel on the right is visible. In the foreground, a 'VisTrails - Spreadsheet - Untitled' window displays a bar chart with blue bars. The x-axis ranges from 1140 to 1180, and the y-axis ranges from 0 to 16. The chart shows a distribution of values with a peak around 1152. The spreadsheet window also includes an 'Export' button and a 'Sheet 1' tab.

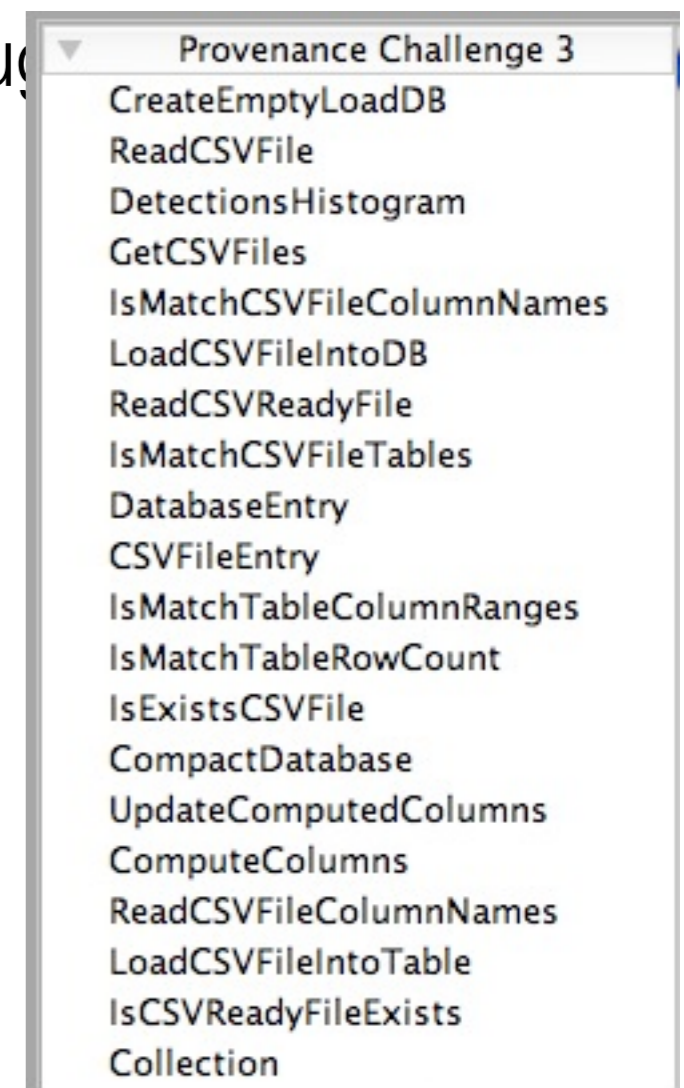
X-axis Value	Y-axis Value (Frequency)
1142	3
1144	5
1146	6
1148	10
1150	13
1152	8
1154	4
1156	10
1158	7
1160	5
1162	8
1164	7
1166	8
1168	9
1170	5
1172	9
1174	6
1176	11
1178	9
1180	8
1182	7
1184	2

Workflow Implementation

- Rewrote LoadAppLogic code python and MySQL
- Wrapped functions as modules and placed in a new VisTrails package; also added new modules that encapsulate multiple fns
- If-error-then-halt steps automatically handled through ModuleErrors raised by the workflow
- Loops, conditionals supported by VisTrails Control Flow package

Workflow Implementation

- Rewrote LoadAppLogic code python and MySQL
- Wrapped functions as modules and placed in a new VisTrails package; also added new modules that encapsulate multiple fns
- If-error-then-halt steps automatically handled through raised by the workflow
- Loops, conditionals supported by VisTrails Control



Workflow Implementation

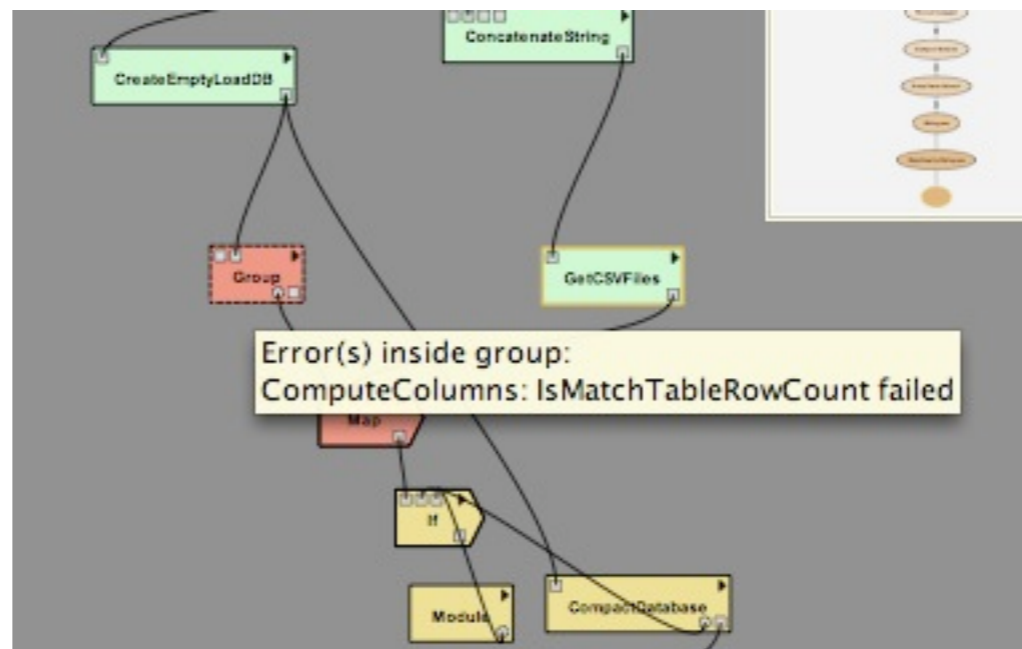
- Rewrote LoadAppLogic code python and MySQL
- Wrapped functions as modules and placed in a new VisTrails package; also added new modules that encapsulate multiple fns
- If-error-then-halt steps automatically handled through ModuleErrors raised by the workflow
- Loops, conditionals supported by VisTrails Control Flow package

Workflow Implementation

- Rewrote LoadAppLogic code python and MySQL
- Wrapped functions as modules and placed in a new VisTrails package; also added new modules that encapsulate multiple fns
- If-error-then-halt steps automatically handled through ModuleErrors raised by the workflow

Workflow Implementation

- Rewrote LoadAppLogic code python and MySQL
- Wrapped functions as modules and placed in a new VisTrails package; also added new modules that encapsulate multiple fns
- If-error-then-halt steps automatically handled through ModuleErrors raised by the workflow



Workflow Implementation

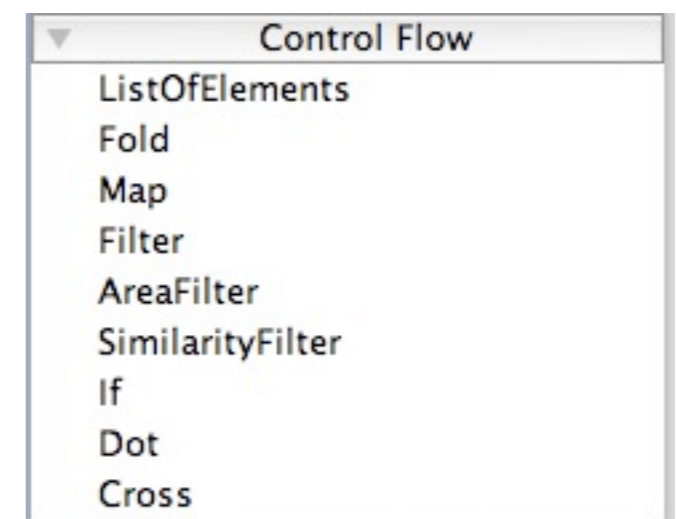
- Rewrote LoadAppLogic code python and MySQL
- Wrapped functions as modules and placed in a new VisTrails package; also added new modules that encapsulate multiple fns
- If-error-then-halt steps automatically handled through ModuleErrors raised by the workflow

Workflow Implementation

- Rewrote LoadAppLogic code python and MySQL
- Wrapped functions as modules and placed in a new VisTrails package; also added new modules that encapsulate multiple fns
- If-error-then-halt steps automatically handled through ModuleErrors raised by the workflow
- Loops, conditionals supported by VisTrails Control Flow package

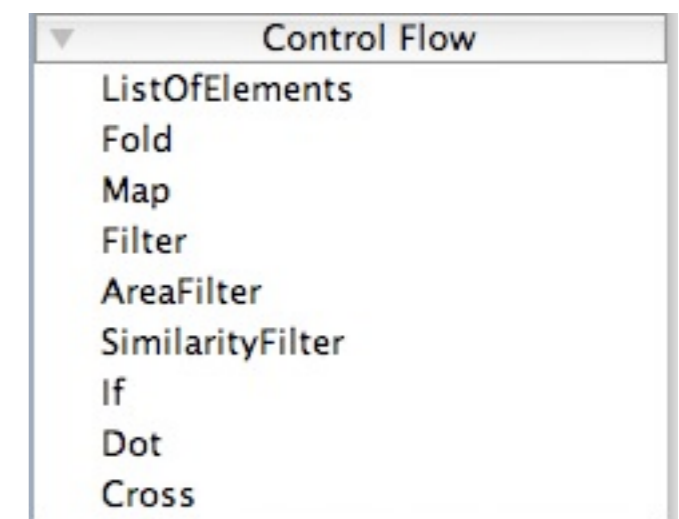
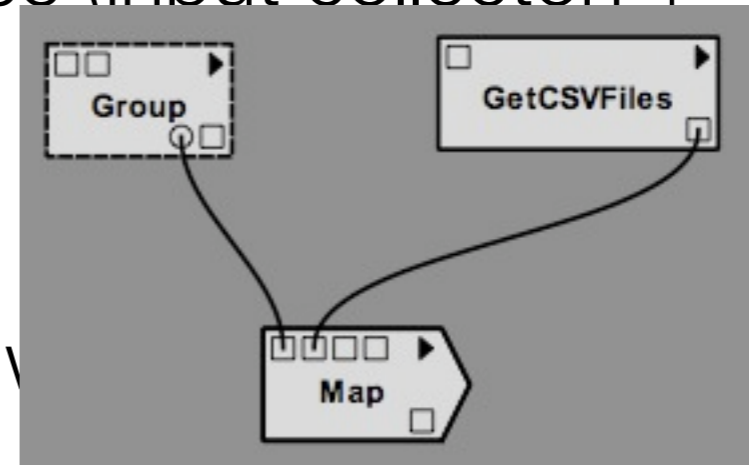
Control Flow

- New package (Fernando Seabra and Rafeal Dahis, UFRJ)
- Fold operations: generalize map-reduce (input collection + subworkflow)
- Combine operation and list of inputs
- Given finite data collections, workflow will not have infinite loops
- Conditionals: two subworkflows as well as a boolean input
- Control flow modules execute subworkflows as part of their own execution



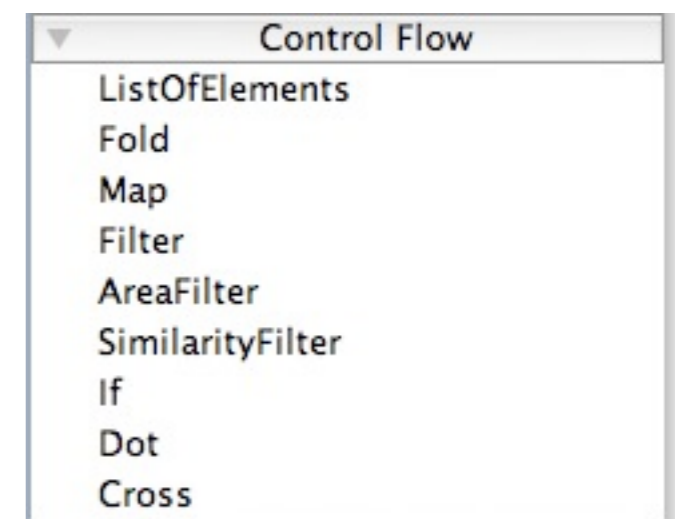
Control Flow

- New package (Fernando Seabra and Rafeal Dahis, UFRJ)
- Fold operations: generalize map-reduce (input collection + subworkflow)
- Combine operation and list of inputs
- Given finite data collections, workflow with loops
- Conditionals: two subworkflows as well as a boolean input
- Control flow modules execute subworkflows as part of their own execution



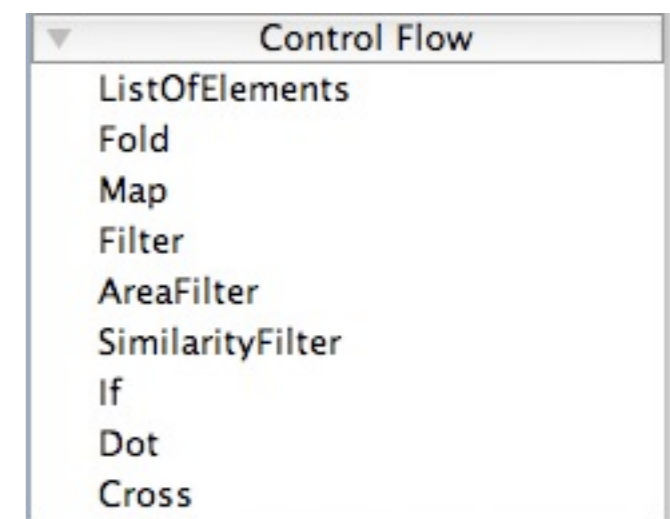
Control Flow

- New package (Fernando Seabra and Rafeal Dahis, UFRJ)
- Fold operations: generalize map-reduce (input collection + subworkflow)
- Combine operation and list of inputs
- Given finite data collections, workflow will not have infinite loops
- Conditionals: two subworkflows as well as a boolean input
- Control flow modules execute subworkflows as part of their own execution



Control Flow

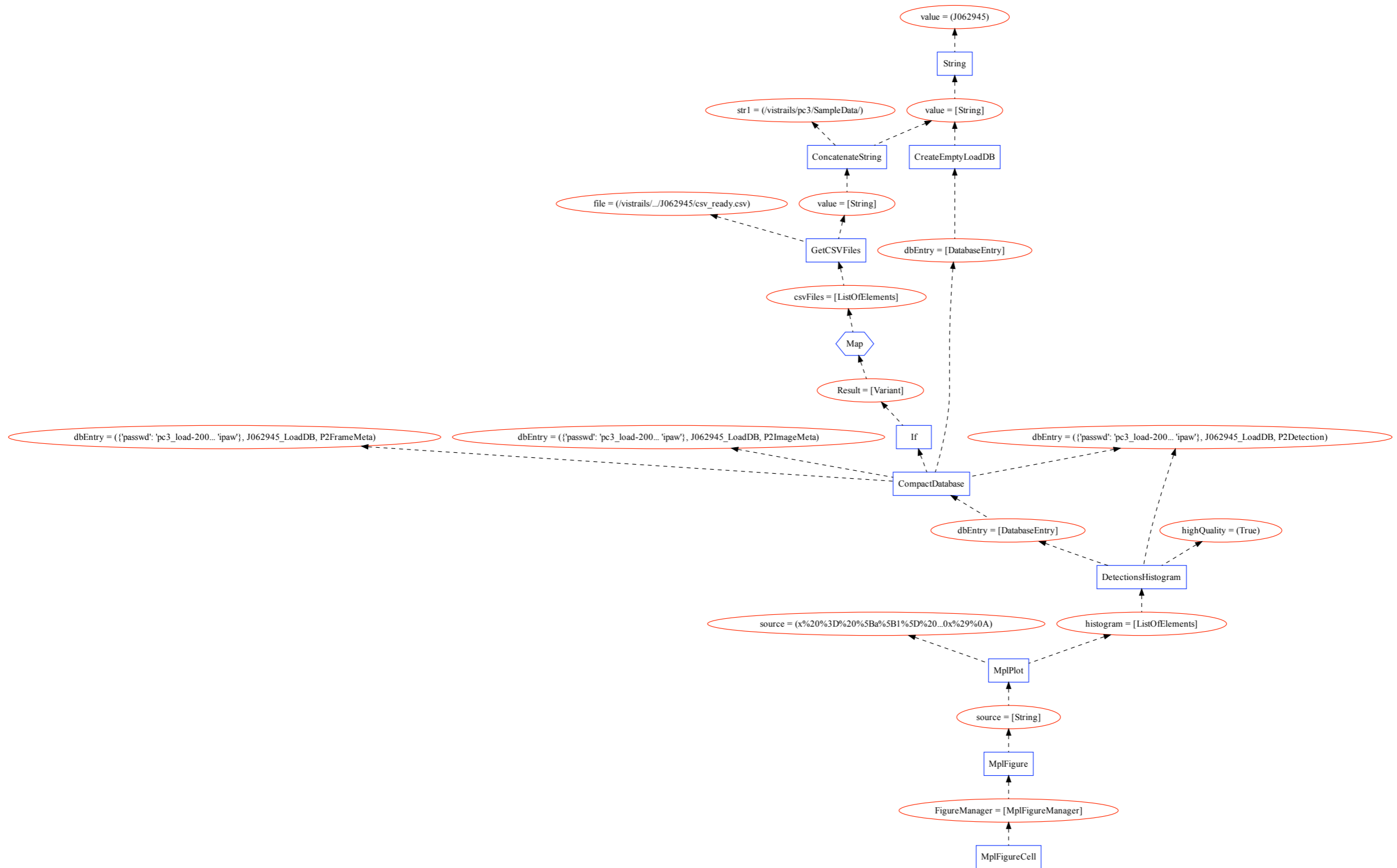
- New package (Fernando Seabra and Rafeal Dahis, UFRJ)
- Fold operations: generalize map-reduce (input collection + subworkflow)
- Combine operation and list of inputs
- Given finite data collections, workflow will not have infinite loops



OPM Implementation

- VisTrails produces layers of provenance: workflow evolution, workflow instantiation, and workflow execution
- Execution information is hierarchical (groups, loops)
- OPM is translated from this provenance by combining information from these layers
- OPM is captured as XML using Southampton's XML Schema
- `value` elements store VisTrails entities

OPM Graph



OPM

- Module executions map to **processes**

```
<process id="p2">
  <value>
    <moduleExec cached="0" completed="1" error="" id="3" machine_id="1"
moduleId="2" moduleName="GetCSVFiles" tsEnd="2009-05-14 16:51:46"
tsStart="2009-05-14 16:51:46">
      <annotation id="1" key="used_files" value="[&apos;/vistrails/pc3/
SampleData/J062945/csv_ready.csv&apos;]" />
    </moduleExec>
  </value>
  <account id="acct0" />
  <account id="acct1" />
  <account id="acct2" />
</process>
```

OPM

- Input Parameters (functions) map to **artifacts**

```
<artifact id="a0">
  <value>
    <function id="4" name="value" pos="0">
      <parameter alias="" id="17" name="&lt;no description&gt;" pos="0"
type="edu.utah.sci.vistrails.basic:String" val="J062945" />
    </function>
  </value>
  <account id="acct0" />
  <account id="acct1" />
  <account id="acct2" />
</artifact>
```

OPM

- Annotations map to **artifacts**

- Save extra information (ie database connection, table names)

```
<artifact id="a35">
  <value>
    <function id="-1" name="dbEntry" pos="0">
      <parameter alias="" id="-1" name="" pos="0"
type="edu.utah.sci.vistrails.basic:String" val="..." />
      <parameter alias="" id="-1" name="" pos="0"
type="edu.utah.sci.vistrails.basic:String" val="J062945_LoadDB" />
      <parameter alias="" id="-1" name="" pos="0"
type="edu.utah.sci.vistrails.basic:String" val="P2Detection" />
    </function>
  </value>
  <account id="acct0" />
  <account id="acct1" />
  <account id="acct2" />
</artifact>
```

OPM

- Connections can be mapped to **artifacts**
 - Transient data: not persisted but exists to show that data was generated and used
 - Output port information captured in provenance

```
<artifact id="a1">  
  <value>  
    <portSpec id="8" name="value" optional="0"  
sigstring="(edu.utah.sci.vistrails.basic:String)" sortKey="-1" type="output" />  
  </value>  
  <account id="acct0" />  
  <account id="acct1" />  
  <account id="acct2" />  
</artifact>
```

OPM

- Subworkflow executions captured as more specific **accounts**

```
<process id="p17">  
  <value>  
    ...  
  </value>  
  <account id="acct2" />  
</process>
```

OPM

- Dependencies are generated from workflow specification
 - module functions connected to modules via **used**
 - modules connected to output ports via **wasGeneratedBy**
 - output ports connected to modules via **used**
 - annotations connected to modules via **used** or **wasGeneratedBy**

OPM Queries

- Querying OPM XML was implemented via XQuery.
- Use three transitivity functions:
 - **derivedFrom**: determine all upstream processes that may have contributed the given artifact
 - **triggeredBy**: determine all upstream processes that may have triggered the given process
 - **triggers**: determine all downstream processes that may have been triggered by the given process
- Use database to determine links between existing data and provenance information about that data

Query Example

```
declare namespace opm='http://openprovenance.org/model/v1.01.a';  
let $d := doc('workflow_opm2.xml')
```

(: The user must find the detection value in the DB. Then we find the table artifact :)
let \$a := \$d//artifact[value/function/parameter/@val = 'P2Detection']

(: return all artifacts upstream containing a P2Detection.csv file :)
return local:derivedFrom(\$d, \$a)[ends-with(value/function/parameter/
@val,'P2Detection.csv')]

Importing OPM

- Have XQuery methods to deal with OPM XML data
- Try to make queries the same across systems
- Tested with SDSC OPM in XML with success
- Changes:
 - Identifying processes, artifacts

Discussion

- Locating OPM entities can be difficult
 - No standard naming to define ids with meaningful names
 - `value` field is not easily searched without understanding the schema of that element
- Queries involved database provenance or attempts to obtain granularity finer than the workflow
- Side effects: database entry is provided as input but the function specification shows no change to the underlying database
- Workflow evolution in OPM