A Brief Introduction to Workflow Generation in the Wings/Pegasus Workflow Framework

www.isi.edu/ikcap/wings

Last Updated: August 11, 2008
THE NEED FOR LAYERS OF ABSTRACTION IN WORKFLOWS AND WORKFLOW SYSTEMS
Similar but not Same: What We Do NOT Mean by “Workflow”

- Workflows in human organizations, eg Process handbook [Malone 95]
  - Task dependencies and flow of information in human organizations

- Workflows that represent process models, eg PSL [Grunninger et al 05]
  - Graphs of activities and subactivities expressing complex temporal and resource constraints

- Workflows for e-business and e-commerce transactions
  - Interactions among processes that support negotiations, multi-party coordinated activities, etc.
Computational vs Service-Based Workflow

- Workflow consists of
  - Nodes: computations
  - Links: dataflow among nodes

**COMPUTATIONAL WORKFLOW**
- Data stored in files
- Metadata catalogs of dataset properties
- Data catalogs of physical replicas
- Components catalogs of code locations and code execution requirements

**SERVICE-BASED WORKFLOW**
- Message-passing paradigm for dataflow
- Service registries show availability
- Standing (3\textsuperscript{rd} party) services
  - Computations managed by service provider
- Services are invoked remotely through potentially complex workflow orchestration
Numerous *Interdependent* Decisions for Workflow Creation

Many types/variants/implementations of application components, each with different storage and computational requirements

Many alternative data collections with different degrees of pre-processing

Many possible data replicas in the distributed environment

Many possible resources to execute application components
Creation of Workflows in Layers of Increasing Detail

1. **Workflow Template** *(generic known-to-work recipes)*
   - Specifies application components and dataflow among them
   - No data specified, just their type

2. **Workflow Instance** *(data-specific)*
   - Specifies data files for a given template
   - Logical file names, not physical file replicas

3. **Executable Workflow** *(actual run)*
   - Specifies physical locations of data files (may be in data repositories)
   - Assigned hosts/pools for execution of each component
   - Includes data movements among execution sites and data repositories as well as data deposition steps
Creation of Workflows in Layers of Increasing Detail

1. **Workflow Template** *(generic known-to-work recipes)*
   - Specifies application components and dataflow among them
   - No data specified, just their type

2. **Workflow Instance** *(data-specific)*
   - Specifies data files for a given template
   - Logical file names, not physical file replicas

3. **Executable Workflow** *(actual run)*
   - Specifies physical locations of data files (may be in data repositories)
   - Assigned hosts/pools for execution of each component
   - Includes data movements among execution sites and data repositories as well as data deposition steps
Workflow Templates: No Data or Execution Information

- Corpus
  - Split
    - 1…n
    - 1…n
  - Kernel_Rules
  - Filter_Rules
    - Prune_Rules
      - Binarize
      - Generate_Rule_Map
- Compile
  - XRS_Rules
  - BRF_Rules
  - Lexicon_Dictionary

Workflow Templates: No Data or Execution Information

- Corpus
  - Split
    - 1…n
    - 1…n
  - Kernel_Rules
  - Filter_Rules
    - Prune_Rules
      - Binarize
      - Generate_Rule_Map
- Compile
  - XRS_Rules
  - BRF_Rules
  - Lexicon_Dictionary
Workflow Instance: Compact Expression

- Corpus
- Split
- Kernel Rules
- XRS_Rules
- BRF_Rules
- Lexicon_Dictionary
- Filter Rules
- Prune Rules
- Generate Rule Map
- Compile

1…n
Workflow Instance: Expanded Expression
Workflow Instance Expressions

- Compact expression for efficient search and matching
- Expanded expression when further details are needed
Executable Workflow: Maps Expanded Instance to Execution Resources
Workflow Creation then Workflow Mapping

Workflow Creation Functions

- Workflow formulation
- Workflow discovery
- Workflow elaboration
- Workflow assembly
- Workflow validation
- Data/parameter selection
- Workflow completion
- Workflow sharing
- Metadata generation

Workflow Mapping and Execution Functions

- Site selection
- Replica selection
- Workflow submission
- Workflow optimization
- Failure recovery
- Data management
WINGS/PEGASUS: WORKFLOW SYSTEM ARCHITECTURE
The Wings/Pegasus Workflow System

[Gil et al 07; Deelman et al 03; Deelman et al 05; Kim et al 08; Gil et al forthcoming]

**WINGS:**
Knowledge-based workflow environment
www.isi.edu/ikcap/wings

- Ontology-based reasoning on workflows and data (W3C’s OWL)
- Workflow library of useful analyses
- Proactive assistance + automation
- Execution-independent workflows

**Pegasus:**
Automated workflow refinement and execution
pegasus.isi.edu

- Optimize for performance, cost, reliability
- Assign execution resources
- Manage execution through DAGMan
- Daily operational use in many domains

**Grid services**
condor.uwisc.edu
www.globus.org

- Secure and controlled sharing of distributed services, computing, data
- Scalable service-oriented architecture
- Commercial quality, open source
Wings: Workflow Instance Generation and Selection

Workflow Selection
- Workflow templates specify complex analyses sequences
- Workflow instances specify data

Workflow Template

Data Selection
- Preexisting data collections
- Workflow execution results

Workflow Instance

WINGS

Workflow Libraries

Ontologies:
 Domain terms,
 Component types,
 Workflow Products
 (OWL)

Application Components
- Specifies data requirements
- Specifies execution requirements

Component Specification

Data Repositories

Executable Workflow

Pegasus

DAGMan/Grid

“Show me workflows that classify datasets”
STUDENT

“Run this workflow with the weather1980 data set”

“Validate this workflow based on the component specs”
SEASONED NL RESEARCHER

“Here is a new classification algorithm, has a parameter for smoothing, is compiled for MPI”
ALGORITHM DEVELOPER
# Workflow System

## Software Overview

<table>
<thead>
<tr>
<th>Workflow System</th>
<th>Workflow Components</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wings</strong></td>
<td>Workflow validation, Data/Comp selection, Workflow generation, Metadata generation</td>
</tr>
<tr>
<td><strong>Pegasus</strong></td>
<td>Site selection, Replica selection, Workflow optimization, Workflow submission</td>
</tr>
<tr>
<td><strong>Condor</strong></td>
<td>DAGMan execution engine, Condor-G job manager, Nagios monitoring probes</td>
</tr>
<tr>
<td><strong>Globus</strong></td>
<td>RLS replica mgmt, GRAM remote submission, GridFTP data transfer</td>
</tr>
</tbody>
</table>

**Legend:**
- All software is open source
- National Middleware Infrastructure (NMI) software

---

© 2005 TANGRAM
Workflow System Uses External Services

Workflow System

Workflow Requests

Results

Workflow Generation
- Component Selection
- Data Selection
- Parameter Selection

Workflow Elaboration & Ranking
- Workflow Elaboration
- Workflow Ranking

Workflow Mapping & Execution
- Workflow Mapping
- Workflow Execution

Workflow Catalog Services
- Workflow Template Catalogs

Metadata Services
- Data Catalogs

Component Services
- Component Catalogs

Provenance Services
- Provenance Catalogs

Execution Services
- Execution Resources
WORKFLOW REPRESENTATION IN WINGS/PEGASUS
Workflow Structure

We take to heart the separation of “programming” from “analysis” activities

- Components are designed by programmers and can be complex (and need testing, debugging, loops should terminate, etc)
- Workflows are composed by non-programmers and should have simple structure-- focus is on selecting application components and data

Therefore, our workflow structure is very streamlined

- Only iterations handled are parallel data processing pipelines
- Only conditionals handled are data-driven component selections
- Standard workflow languages offer much more complex constructs

Workflow structure designed to:

- Be accessible to users
- Facilitate automation and failure recovery
Blocks and Arrows A Workflow Do Not Make: A Well-Formed Workflow Sketch

- Components are models of software to be executed
- Components have all I/O data exposed
- All important parameter settings in components are exposed
  - So workflow system can reflect settings in provenance records
- Links reflect ALL data flow
  - Each dataset is produced by one component, can be consumed by several
- No side effects
  - No global context
Simple Examples of Workflows for Machine Learning Experiments

**Very specific, limited applicability**

```
ClassIndex = 5
```

**Very general, very broad applicability**

```
Training Data → ID3-MODELER → Model
Test Data → ID3-CLASIFIER → Classification
```

```
Training Data → DISCRETIZE → SAMPLE → BAYES-MODELER → Model
Test Data → BAYES-CLASIFIER → Classification
```

**General but customized to a specific use**

```
Sampling Interval = 20
```

```
Weather test data → ID3-MODELER → Weather-SM-2007-Model.csv
ClassIndex = 5
```

```
Weather test data → ID3-CLASIFIER → Weather prediction
```
Representing Workflows

- Workflow dataflow has graphic rendering
- Workflow constraints:
  TrainingDataVariable ≠ TestDataVariable
  Domain of TrainingDataVariable = Domain of TestDataVariable
Representing Workflows: Another Example

- **Workflow Constraints:**
  - `TestDataVariable` has Domain = `weather`
  - Training DataVariable = Weather -SM-2007-Data.csv
  - `SamplingIntervalParameterVar` = 20
  - `ClassIndexParameterVar` = 5
  - TrainingDataVariable ≠ TestDataVariable
  - Domain of TrainingDataVariable = Domain of TestDataVariable
AUTOMATIC WORKFLOW GENERATION IN WINGS
Workflow Generation Algorithm

- **Input:** Workflow request
  - Needs to specify:
    - High-level workflow template to be used
    - Additional constraints on template data variables
  - Does not need to specify:
    - Data sources to be used
    - Particular component to be used
    - Parameter settings for the components

- **Output:** workflow instances ready to submit to Pegasus for mapping and execution

- **How it works:** the algorithm automatically finds appropriate data sources, components, and parameter settings appropriate for the request
  - Maintains a pool of possible workflow candidates, each algorithm step elaborates existing candidates and may eliminate or add candidates in the process
Automatic Template-Based Workflow Generation Algorithm

Algorithm: TEMPLATE-BASED-WORKFLOW-GENERATION

Input: request
Output: workflow-instances

Seed Workflows from Request
1. seeded-workflows ← {} for each template-seed-pair ∈ request do 2. workflows ← SEED-WORKFLOW(template-seed-pair) 3. if (workflows ≠ null) then 4. seeded-workflows ← seeded-workflows ∪ workflows 5. if (seeded-workflows = null) then workflow-instances ← {}; return

Find Input Data Requirements
7. binding-ready-workflows ← {} for each seeded-workflow ∈ seeded-workflows do 8. workflows ← BACKWARD-SWEEP(seeded-workflow) 9. if (workflows ≠ null) then 10. binding-ready-workflows ← binding-ready-workflows ∪ workflows 11. if (binding-ready-workflows = null) then workflow-instances ← {}; return

Data Source Selection
13. bound-workflows ← {} for each binding-ready-workflow ∈ binding-ready-workflows do 14. workflows ← SELECT-INPUT-DATA-OBJECTS(binding-ready-workflow) 15. if (workflows ≠ null) then 16. bound-workflows ← bound-workflows ∪ workflows 17. if (bound-workflows = null) then workflow-instances ← {}; return

Parameter Selection
19. configured-workflows ← {} for each bound-workflow ∈ bound-workflows do 20. workflows ← FORWARD-SWEEP(bound-workflow) 21. if (workflows ≠ null) then 22. configured-workflows ← configured-workflows ∪ workflows 23. if (configured-workflows = null) then workflow-instances ← {}; return

Workflow Ranking

Workflow Instantiation and Grounding
29. workflow-instances ← {} for each ranked-workflow ∈ ranked-workflows do 30. workflow ← INSTANTIATE-WORKFLOW(ranked-workflow) 31. workflow-instances ← workflow-instances ∪ {workflow} 32. ground-workflows ← GROUND-WORKFLOW(workflow) 33. return ground-workflows
Workflow request = Workflow Template + Seed Constraints

Seed constraints:
- modelerClassIndex wflow:hasParameterValue 5
- modelerOutputModelDataVariable dcdm:hasDomain dcdm:weather
- randomSampleNClassIndex wflow:hasParameterValue 5
- discretizeClassIndex wflow:hasParameterValue 5
Step 1: Workflow Template is Seeded

Seed workflow from request → seeded workflows

Find input data requirements → binding-ready workflows

Data source selection → bound workflows

Parameter selection → configured workflows

Workflow instantiation → workflow instances

Workflow grounding → ground workflows

Workflow ranking → top-k workflows

Workflow mapping → executable workflows

unified well-formed request
Step 2: Backward Sweep

Seed workflow from request

Find input data requirements

Data source selection

Parameter selection

Workflow instantiation

Workflow grounding

Workflow ranking

Workflow mapping

unified well-formed request

seeded workflows

binding-ready workflows

bound workflows

configured workflows

workflow instances

ground workflows

top-k workflows

executable workflows
Step 3: Select Data Sources

- Seed workflow from request
- Find input data requirements
- Data source selection
- Parameter selection
- Workflow instantiation
- Workflow grounding
- Workflow ranking
- Workflow mapping

unified well-formed request
seeded workflows
binding-ready workflows
bound workflows
configured workflows
workflow instances
ground workflows
top-k workflows
executable workflows

E-07
S-NY

Yolanda Gil (g)
Step 3: Select Data Sources

1. Seed workflow from request
2. Find input data requirements
3. Data source selection
   - binding-ready workflows
4. Parameter selection
   - configured workflows
5. Workflow instantiation
   - workflow instances
6. Workflow grounding
   - ground workflows
7. Workflow ranking
   - top-k workflows
8. Workflow mapping
   - executable workflows

Unified well-formed request

Seeded workflows

Binding-ready workflows

Configured workflows

Workflow instances

Ground workflows

Top-k workflows

Executable workflows
Step 4: Forward Sweep

1. Seed workflow from request
2. Find input data requirements
3. Data source selection
4. Parameter selection
5. Workflow instantiation
6. Workflow grounding
7. Workflow mapping
8. Workflow ranking
9. Top-k workflows
10. Workflow instances
11. Ground workflows
12. Configured workflows
13. Binding-ready workflows
14. Seeded workflows
15. Unified well-formed request

E-07

S-NY

Yolanda Gil (g)
Step 5: Workflow Instantiation

- Seed workflow from request
- Find input data requirements
- Data source selection
- Parameter selection
- Workflow instantiation
- Workflow grounding
- Workflow ranking
- Workflow mapping

unified well-formed request
seeded workflows
binding-ready workflows
bound workflows
configured workflows
workflow instances
ground workflows
top-k workflows
executable workflows
Step 5: Workflow Instantiation

1. Seed workflow from request
2. Find input data requirements
3. Data source selection
4. Parameter selection
5. Workflow instantiation
   - unified well-formed request
   - seeded workflows
   - binding-ready workflows
   - bound workflows
   - configured workflows
   - workflow instances
   - ground workflows
   - top-k workflows

Workflow grounding

Workflow ranking

Workflow mapping

Result-PartA

Result-PartB

E-07

S-NY
Step 6: Workflow Grounding

1. Seed workflow from request
2. Find input data requirements
3. Data source selection
4. Parameter selection
5. Workflow instantiation
6. Workflow grounding
7. Workflow ranking
8. Workflow mapping

- unified well-formed request
- seeded workflows
- binding-ready workflows
- bound workflows
- configured workflows
- workflow instances
- ground workflows
- ground workflows
- top-k workflows
- executable workflows

Ground Workflow

<job id = "j42" name="Neuman-BC">
<argument> -i E-07 17.5 -o ES-07....

Yolanda Gil (g)
Step 7: Workflow Ranking

Seed workflow from request

Find input data requirements

Data source selection

Parameter selection

Workflow instantiation

Workflow grounding

Workflow ranking

Workflow mapping

unified well-formed request

seeded workflows

binding-ready workflows

bound workflows

configured workflows

workflow instances

ground workflows

top-k workflows

executable workflows

W1: estimated exec time 3hrs

W2: estimated exec time 20hrs

W3: estimated exec time 3d

W4: estimated exec time 5hrs
Step 7: Workflow Ranking

- Seed workflow from request
- Find input data requirements
- Data source selection
- Parameter selection
- Workflow instantiation
- Workflow grounding
- Workflow ranking

- unified well-formed request
- seeded workflows
- binding-ready workflows
- bound workflows
- configured workflows
- workflow instances
- ground workflows
- top-k workflows

W1: estimated exec time 3hrs
W2: estimated exec time 20hrs
W3: estimated exec time 3d
W4: estimated exec time 5hrs

Top-k workflows: W1, W4

Bottom-k workflows: W2, W3
Step 8: Workflow Mapping

Ground workflow: 15 compute nodes devoid of resource assignment

Executable workflow: mapped to 3 sites

- 13 data stage-in nodes
- 11 compute nodes (1-2&5-6 reduced based on available intermediate data)
- 8 inter-site data transfers
- 14 data stage-out nodes to long-term storage
- 14 data registration nodes (data cataloging)
Why Do We Automate All This? So You Don’t Have To

[Diagram showing workflow from request to executable workflows]

Seed workflow from request
Find input data requirements
Data source selection
Parameter selection
Workflow instantiation
Workflow grounding
Workflow mapping

<table>
<thead>
<tr>
<th>Request ID</th>
<th>Workflow candidates generated + considered (many are eliminated)</th>
<th>Queries about data</th>
<th>Queries about tools</th>
<th>Workflow Generation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>6 8 8</td>
<td>1 6 8</td>
<td>5 s</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td>6 8 8</td>
<td>7 6 16</td>
<td>4 s</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>6 24 24</td>
<td>7 6 48</td>
<td>7 s</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>6 24 24</td>
<td>13 6 72</td>
<td>8 s</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>18 64 48</td>
<td>7 18 128</td>
<td>22 s</td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td>18 286 216</td>
<td>7 18 576</td>
<td>81 s</td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td>18 16 12</td>
<td>7 18 32</td>
<td>10 s</td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td>6 0 0</td>
<td>1 6 0</td>
<td>1 s</td>
<td></td>
</tr>
</tbody>
</table>
EXTERNAL SERVICES: ASSUMPTIONS
Workflow System Uses External Services

Workflow Requests

Workflow System

Workflow Generation
- Component Selection
- Data Selection
- Parameter Selection

Workflow Elaboration & Ranking
- Workflow Elaboration
- Workflow Ranking

Workflow Mapping & Execution
- Workflow Mapping
- Workflow Execution

Workflow Catalog Services
- Workflow Template Catalogs

Metadata Services
- Data Catalogs

Component Services
- Component Catalogs

Provenance Services
- Provenance Catalogs

Execution Services
- Execution Resources

Results
### Representing Components (I)

**OWL representations of:**
- Component classes
- Component arguments: inputs, outputs and parameters
- Constraints in component arguments

<table>
<thead>
<tr>
<th>(a) Abstract Component</th>
<th>(b) Concrete Components</th>
</tr>
</thead>
</table>
| **Component ID:** DecTreeModeler  
is Abstract  
Input: d: Dataset  
hasSize s:Size  
Params: i: ClassIndex  
hj: maxJavaHeapSize  
  h j <- 256x rem(s/1000)  
Output: mo: Model is DecTree |
| **Component ID:** ID3-Modeler  
is DecTreeModeler  
is Concrete  
Input: d: Dataset is **Discrete**  
Params: i: ClassIndex  
hj: maxJavaHeapSize  
Output: mo: DecTree-Model |
| **Component ID:** Lmt-Modeler  
is DecTreeModeler  
is Concrete  
Input: d: Dataset is **NoMissingVals**  
Params: i: ClassIndex  
hj: maxJavaHeapSize  
Output: mo: DecTree-Model |
Representing Components (II)

- Backward sweeping rules: given argument constraints, what are the input argument properties required

```prolog
// number of classes is propagated backward by a Classifier
[classifierTransferNClass:
  (?c rdf:type pcdom:Classifier)
  (?c pc:hasOutput ?odv) (?odv ac:hasArgumentID "o")
  (?c pc:hasInput ?idvmodel) (?idvmodel ac:hasArgumentID "m")
  (?c pc:hasInput ?idvdata) (?idvdata ac:hasArgumentID "d")
  (?odv dcdom:hasNumberOfClasses ?val)
  -> (?idvmodel dcdom:hasNumberOfClasses ?val),
     (?idvdata dcdom:hasNumberOfClasses ?val)]
```

- Forward sweeping rules: given argument constraints, what are the output argument properties required

```prolog
(?c rdf:type pcdom:Modeler)
  (?c pc:hasInput ?idv) (?idv ac:hasArgumentID "d")
  (?c pc:hasInput ?ipv) (?ipv ac:hasArgumentID "j") // maxJavaHeapSize
  (?idv dcdom:hasNumberOfInstances ?x) lessThan(?x 1000)
  -> (?ipv ac:hasValue "256M")
```
# Calls to Component Services and Purpose

<table>
<thead>
<tr>
<th>Function in Component Services</th>
<th>Purpose in Automatic Generation Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>c:inputs</td>
<td>Validate workflow template in request</td>
</tr>
<tr>
<td>c:parameters</td>
<td>Ground workflows to be submitted for execution</td>
</tr>
<tr>
<td>c:outputs</td>
<td>Use of abstract components in workflow templates that can be specialized in backward and forward sweep</td>
</tr>
<tr>
<td>c:args</td>
<td>Generate binding-ready workflows in backward sweep</td>
</tr>
<tr>
<td>c:invocation-command</td>
<td>Generate configured workflows in forward sweep</td>
</tr>
<tr>
<td>c:is-concrete</td>
<td></td>
</tr>
<tr>
<td>c:specialize</td>
<td></td>
</tr>
<tr>
<td>c:specialize-to-concrete</td>
<td></td>
</tr>
<tr>
<td>c:is-backward-enabled</td>
<td></td>
</tr>
<tr>
<td>c:find-DODs-given-output-requirements</td>
<td></td>
</tr>
<tr>
<td>c:is-forward-enabled</td>
<td></td>
</tr>
<tr>
<td>c:predict-DODs-given-input-requirements</td>
<td></td>
</tr>
<tr>
<td>c:is-configurable</td>
<td></td>
</tr>
<tr>
<td>c:configure</td>
<td></td>
</tr>
<tr>
<td>c:is-configured</td>
<td></td>
</tr>
<tr>
<td>c:estimate-performance</td>
<td>Rank candidate workflows</td>
</tr>
</tbody>
</table>
## API to Medatada Services and Purpose

<table>
<thead>
<tr>
<th>Function in Metadata Services</th>
<th>Purpose in Automatic Generation Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>d:combine-DODs</td>
<td>Seed workflow templates</td>
</tr>
<tr>
<td>d:entity-DODs</td>
<td>Filter relevant data properties to be propagated in the workflow enables workflow candidate addition and elimination</td>
</tr>
<tr>
<td>d:assign-identifier</td>
<td>Create unique identifiers and properties for workflow data products so they can be reused in future workflows</td>
</tr>
<tr>
<td>d:assert-predicted-DODs</td>
<td>Selection of input data enables creation of bound workflows</td>
</tr>
<tr>
<td>d:find-data-objects</td>
<td>Propagation of input data properties in forward sweep enables component specialization and workflow candidate elimination</td>
</tr>
<tr>
<td>d:obtain-DODs</td>
<td></td>
</tr>
</tbody>
</table>
WORKFLOWS FOR THE SOUTHERN CALIFORNIA EARTHQUAKE CENTER (SCEC)
Seismic Hazard Analysis in Southern California Earthquake Center (SCEC) [Slide from T. Jordan]

- **Seismicity**
- **Paleoseismology**
- **Local site effects**
- **Geologic structure**

- **Faults**
- **Stress transfer**
- **Crustal motion**
- **Crustal deformation**
- **Seismic velocity structure**

- **Seismic Hazard Model**

InSAR Image of the Hector Mine Earthquake

- A satellite generated Interferometric Synthetic Radar (InSAR) image of the 1999 Hector Mine earthquake.
- Shows the displacement field in the direction of radar imaging.
- Each fringe (e.g., from red to red) corresponds to a few centimeters of displacement.
Reusable High-Level Workflow Templates

Intensional descriptions of data sets

Intensional descriptions of parallel computations

Querying results of other data creation subworkflows

Rich metadata descriptions for all data products
Propagagation of metadata for filename generation: an example
DAX automatically generated from WINGS

14,639 jobs for 4,626 ruptures with 106,124 rupture variations for USC site
Workflows for Seismic Hazard Analysis

[Gil et al 06; Kim et al 06; Gil et al 07]

- Input data: a site and an earthquake forecast model
  - thousands of possible fault ruptures and rupture variations, each a file, unevenly distributed
  - ~110,000 rupture variations to be simulated for a given site

- High-level template combines 11 application codes

- 8048 application nodes in the workflow instance generated by Wings

- 24,135 nodes in the executable workflow generated by Pegasus, including:
  - data stage-in jobs, data stage-out jobs, data registration jobs

- Executed in USC HPCC cluster, 1820 nodes w/dual processors) but only < 144 available
  - Including MPI jobs, each runs on hundreds of processors for 25-33 hours
  - Runtime was 1.9 CPU years

- Provenance records kept throughout the generation and execution process for 100,000 workflow data products
WORKFLOWS FOR MICROSCOPY IMAGE CORRECTION
(with Saltz et al at OSU)
Microscopy Image Correction [Saltz et al 05]

- Microscope takes image tiles, one at a time, for all the slices
  - tens of GBytes uncompressed

- Image is corrected, then warped to map to standard brain, then pre-processed for efficient querying

- Correction workflow: original image has repeated areas across tiles, this needs to be corrected. Steps:
  1. Z-projection: picks the best pixel from all the z-slices.
  2. Normalization
  3. Auto-alignment: piece together the image mosaic from the tiles
     - Uses a feature matching algorithm, output is "offsets", the alignment is given a score, goal is to maximize the score
  4. Stitching: remove overlapping regions, so the tiles can "transition" more smoothly by finding best match across tiles.
     - Optimization algorithm picks the right offsets and stitches them together
Workflow Template

Sketch

Format conversion

Controllable parameter: \( p \) is number of chunks in an image layer

Set of z layers each with t tiles (.IMG)

Set of z layers each with t tiles (.DIM)

Chunksize

Set of stacks of chunks
Collections of z layers, each has \( p \) chunks, each chunk has t tiles)

Z-projection

\( p \) (one per chunk stack)

Sets of chunks for z-proj

Generate magic tiles

Internally iterates over each chunk of the z-proj

2 magic tiles

Normalize

\( p \) (one per chunk of z-proj)

Sets of normalized chunks for z-proj

Auto-align

Internally iterates over each chunk of the normalized z-proj

Scores per pairs of chunks in z-proj

MST

Graph

Normalize

\( p \times z \) (one per chunk per layer)

Sets of normalized chunks for all layers

Stitch

Internally iterates over each chunk of each layer

Sets of stitched norm. chunks for all layers

Legend

- Regular codes
- Parallel codes (MPI) with one computation node
- Will be expanded to several nodes

Set of z layers each with t tiles (.IMG)

Set of z layers each with t tiles (.DIM)

Set of stacks of chunks
Collections of z layers, each has \( p \) chunks, each chunk has t tiles)

Sets of chunks for z-proj

Internally iterates over each chunk of the z-proj

2 magic tiles

Sets of normalized chunks for z-proj

Internally iterates over each chunk of the normalized z-proj

Scores per pairs of chunks in z-proj

Graph

Sets of stitched norm. chunks for all layers
Sketch of Workflow Template

Controllable parameter: $p$ is number of chunks in an image layer

Format conversion

Chunksize

Z-projection

Generate magic tiles

Normalize

Auto-align

MST

Normalize

Stitch

Will be expanded to several computation nodes

Parallel codes (MPI) with one computation node

Regular codes

LEGEND
Sketch of computations in the workflow:

- **Format conversion**
- **Chunkisize**
  - **Z-proj**
  - **Z-proj**
  - **Z-proj**
  - **p**
- **Generate magic tiles**
  - **Norm**
  - **Norm**
  - **Norm**
  - **p**
- **Auto-align**
- **MST**
- **Norm**
  - **Norm**
  - **Norm**
  - **Norm**
  - **p * z**
- **Stitch**

**Legend**:
- Regular codes
- Parallel codes (MPI) with one computation node
- Computation nodes

Controllable parameter: p is number of chunks in an image layer.
SUMMARY
Summary: Creating Workflows with WINGS

- Separates analysis spec from data
  - Workflow template as reusable well-defined acceptable analysis process
  - Workflow instance binds template to data for particular analyses
- Ensures that the data complies with the component specifications and their constraints within the workflow
- Represents data collections (nominal or otherwise) within the workflow specification
- Automatically generates descriptions and metadata to new data products to be created by the workflow execution
- Compact workflow instance is user-friendly and reusable
  - Separates data provenance (workflow instance) and pedigree (workflow template)
- Expands workflow instance into DAX for Pegasus, which creates the executable workflow
Key Benefits

- Efficient and correct creation of new workflows
  - By retrieving a template and filling in the data

- Framework ensures adherence to methodology
  - Represents as templates widely-accepted analysis methodologies
  - Supports repeatability of experiments/analyses
  - Enables controlled variations

- Ensures better quality of data analysis results
  - Attaches provenance and pedigree information
Ongoing and Future Work

- Interactive assistance in creating valid workflow templates
  - Based on CAT (Composition Analysis Tool) [Kim et al 05]
- More sophisticated models of components
- Automatic completion of workflow’s data conversion and formatting steps through AI planning techniques
- Tracking new versions of components, invalidate data and workflows from old versions
- Workflow template libraries
  - Indexing, retrieval
- Managing collections of workflows as part of an overall analysis activity
  - Eg: parameter sweeping, variants of analysis