Methods and Tools for Development and Running Collaborative Applications

Marian Bubak, Tomasz Gubała and Maciej Malawski
Institute of Computer Science & ACC CYFRONET
AGH University of Science and Technology
Krakow, Poland
bubak@agh.edu.pl
An approach to scientific investigations which, besides of analyses of individual phenomena, integrates different, interdisciplinary sources of knowledge about a complex system, to acquire understanding of the system as a whole.

Outline

• Motivation – applications, requirements
• Idea of an “experiment”
• Method – scripts, grid object abstraction
• Examples of experiments
• Virtual laboratory architecture
• Modules - tools
• Summary
ViroLab Users
Required Functionality

- **Development:**
  - focus on computational functionality
  - easy access to remote data

- **Experiment sharing:**
  - creation of experiments by teams of developers

- **Browsing experiments:**
  - web application for browsing and executing experiments

- **Running experiments:**
  - single-click experiment execution
  - interactive communication between experiment and user

- **Gathering results:**
  - dedicated renderers for script input and output parameters
  - provenance storage and searching

- **Feedback:**
  - easy communication between end users and developers
On top of any infrastructure ...

- **Users**
  - Experiment developer
  - Scientist
  - Clinical Virologist

- **Interfaces**
  - Experiment Planning Environment
  - Experiment scenario
  - ViroLab Portal
  - Patient Treatment Support

- **Runtime**
  - Virtual Laboratory runtime components
    - (Required to select resources and execute experiment scenarios)

- **Services**
  - Computational services
    - (services (WS, WTS, WS-R))
    - components (MOCCA)
    - jobs (EGEE, AHE)
  - Data services
    - (DAS data sources, standalone databases)

- **Infrastructure**
  - Grids, Clusters, Computers, Network
Experiment

- **Experiment** - a process that combines together data with a set of activities that act on that data
Experiment Lifecycle

• *Experiment Pipeline* - is a collaborative planning and execution process that may create a new experiment
Workflow Systems and Virtual Laboratories

<table>
<thead>
<tr>
<th>Virtual Laboratory</th>
<th>App construction</th>
<th>Middleware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kepler</td>
<td>Drag&amp;Drop</td>
<td>Ptolemy II, Globus Toolkit, WS</td>
</tr>
<tr>
<td>Triana</td>
<td>Drag&amp;Drop</td>
<td>Globus Toolkit, GAT</td>
</tr>
<tr>
<td>myGrid, Taverna</td>
<td>Drag&amp;Drop</td>
<td>Globus Toolkit, WS</td>
</tr>
<tr>
<td>Geodise</td>
<td>Matlab scripts</td>
<td>Computational Toolbox, Jython, WS, Java CoG (1.1), GT2</td>
</tr>
<tr>
<td>NESSgrid</td>
<td>webpage with tools</td>
<td>Globus Toolkit, WS</td>
</tr>
<tr>
<td>VL-e</td>
<td>Drag&amp;Drop</td>
<td>Globus Toolkit, SoapLab</td>
</tr>
<tr>
<td>VL PSNC</td>
<td>JWS apps, batch jobs</td>
<td>Globus Toolkit, GAT</td>
</tr>
</tbody>
</table>

Conclusions:
- There is no solution that fulfills all requirements
- Many useful ideas: semantic modeling, tool registry, provenance tracking etc.
- Using a scripting language - useful (Geodise)
High-level Script Approach

• Application composition based on dynamic scripting language
• Rapid application development, prototyping, experiments (scientific applications)
• High-level API
  – Rich functionality (standard library), few lines of code needed
  – Easy to learn, clear and readable code
  – Supports full set of control structures (high expressiveness)
• Candidate languages
  – Python, Ruby, Lua, Perl …
• Solution: JRuby
  – Object oriented, clear syntax, dynamic
  – Integration with Java (important for use of existing technologies)

Uniform interface to computational resources
Grid Object Abstraction

- **Grid Object Class** – a group that has similar functionality with regard to its domain operations
- **Grid Operation** – specification of an activity with descriptions of input parameters and output results
- **Grid Object Implementation** – realization of Grid Object Class in concrete technology (e.g. WS, MOCCA)
- **Grid Object Instance** – an implementation that is deployed and can be accessed by the Invoker
Virus Genotype Analysis

**Objective:** loads nucleotide sequence of an HIV virus strain and provides its mutations and drug ranking information

- Gems used:
  - Alignment
  - Subtype detection
  - Drug ranking
  - Data Access Service

- Input: virus nucleotide sequence

- Output: various analyses

http://virolab.cyfronet.pl/trac/vlvl/wiki/ExperimentDemo
Experiment Plan

patientID = 6  region = "rt"

remoteDB = DACConnector.new( "DAS","virolab.hlrs.de")
sequences = remoteDB.executeQuery("select nucleotides from nt_sequence where patient_ii=#{patientID.to_s};")
regaDBMutationsTool = GObj.create('regadb.RegaDBMutationsTool')
regaDBMutationsTool.align(sequences, region.upcase)
mutations = regaDBMutationsTool.getResult
regaDBSubtypingTool = GObj.create('regadb.RegaDBSubtypingTool')
regaDBSubtypingTool.subtype(sequences[0])
puts regaDBSubtypingTool.getResult
puts drs.drs('retrogram', region, 100, mutations)
Collaborative Applications on VLvl

Experiment Repository

Scientist

PROToS Provenance System

GSEngine

Data Access Client

rdb = DACConnector.new(
  "mysql","virolab.cyfornet.pl",
  "test","testuser",""
)
mutation = rdb.executeQuery(
  "select mutation from patients
   where patient_id = patientID;"
)

Drugs System

dr_system = GObj.create(
  'org.virolab.DrugRankingSystem'
)

Invoke operation

result = dr_system.drs(
  'retrogram', region, 100, mutation)

Data sources

Remote Relational Data Base

Data Access Service
Protein Folding

Objective:
- demonstrate the usage of Virtual Laboratory for proteomics applications

- Input: protein and chain ID
- Output: 3D structure of protein
- Gems used:
  - Protein Data Bank (PDB) Web Service
  - Early-stage protein folding
  - DAC and WebDAV for result storage

http://virolab.cyfronet.pl/trac/exampleExperiments/wiki/exex/Folding
Data Mining with Weka

**Objective:** to analyze the quality of various classification algorithms on large datasets using Weka data mining library and MOCCA component framework.

- Input: sample dataset
- Output: quality of predictions
- Gems used:
  - Web services for data retrieval, conversion, splitting and testing
  - MOCCA components wrapping algorithms from Weka
  - WebDAV server for data storage

http://virolab.cyfronet.pl/trac/exampleExperiments/wiki/exex/WekaAdv
Virtual Laboratory Architecture

Experiment Repository

Experiment Management Interface

Experiment Planning Environment

Presentation

ViroLab Portal

Decision Support Portlet

QUaTRO Portlet

Domain Ontology Store

Data Access

- Data source access
- Monitoring and messaging
- Encryption and decryption
- Access authorization
- Response transformation

Collaboration Tools

Interaction

Grid Resources Registry

Experiment execution

Grid Object information

Data Access Client

GridSpace Engine

- JRuby interpreter
- Runtime libraries
- Optimizer

Grid Operation Invoker

Search

Search

Provenance queries

Resource state

Events regarding provenance

Execution events, resource state

Invocation

Middleware

PROToS/QUaTRO

Monitoring Infrastructure

Execution monitoring information
Experiment Planning Environment

**Objective:** to facilitate the experiment development process by providing an integrated and collaborative environment

- Assist in the development of experiment plans powerful GScript editor
- Execute experiments by integrating with the GSEngine runtime system
- Support collaboration between VL users groups by:
  - Sharing experiment within groups of experiment developers
  - Releasing experiments for scientific users
  - Collecting feedback from experiment users
- Enable extendability with new features (e.g. GRR and Onto Browser plugins)
Objective: to provide ViroLab users with an easily accessible and convenient facility for experiment and result management

• Browse experiment repositories in search for a suitable experiment plan
• Interactively run any number of experiment plans and monitor their execution status
• Retrieve, analyze and store scientific results
• Extensible support for different security models with full Shibboleth integration
Domain Ontology Store

**Objective:** To base the integration between middleware components and user interfaces on domain-related taxonomies and concepts.

- Domain knowledge is modeled with OWL-transcribed models.
- Models are stored in a secure, persistent store with an open access protocol.
- Other components and tools are able to query the store using several query languages.
- Ontology Browser Plugin for experiment developers to use their domain concepts when planning new experiments.
Objective: to store and version Experiment Plans, which can be executed by the GSEngine interpreter

- Experiment Plans arranged in a specific structure
- Experiment Repository Client – provides access to Experiment Repositories
  - Facilitates releasing Experiment Plans to the repository from EPE
  - Enables downloading Experiment Plans for execution by EMI and the GSEngine Server interpreter
  - Enables managing feedback in EMI
- Pluggable adapters for different version control systems
Grid Resource Registry

**Objective:** To store information about all available and accessible computational resources, hides the technological complexity and provides uniform and user-friendly access to web services, component and grid infrastructures

- Two layers of resource description
  - technology-independent
  - technology-specific

- ResBrowser Plugin assists developers during experiment development

- GRRAdmin Plugin allows managing information stored inside the registry
**GridSpace Engine**

**Objective:** To provide a Virtual Laboratory engine of for running experiments and an entry point for other capabilities of Virtual Laboratory

- Environment for running experiments
  - Interpreter of GScript based on the Ruby programming language
  - Contains libraries dedicated to the Virtual Laboratory written in Ruby or Java
  - Experiments provided with environmental context (security, monitoring, etc.)

- Entry point for the Virtual Laboratory
  - Proxying Data Access Service
  - Running experiments stored within Experiment Repository
  - Integration with monitoring infrastructure and provenance
Objective: to provide developers of virtual experiments with a uniform interface for computational resources that enables transparent and coherent usage of web services, components and grid infrastructures, thus allowing integration of tools exposed with diverse middleware technologies in an easy manner.

- Adapters for various middleware technologies
- Simple and clear developer API
- Full integration with GScript syntax
- Usable as a standalone component
Objective: to harness diverse computation models and technologies (Web Services, components, job submission – grid systems) and make them available in the Virtual Laboratory in a convenient way

- Pluggable Grid Operation Invoker adapters for supported technologies
- Development and experiments with component-based middleware: MOCCA
  - Component composition
  - Inter-framework interoperability
- Security for component-based middleware
- Monitoring of component- and service-oriented middleware
# Summary (1)

<table>
<thead>
<tr>
<th>Features</th>
<th>Scripts</th>
<th>Workflows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical constructs</td>
<td>classes, methods, blocks</td>
<td>subworkflows</td>
</tr>
<tr>
<td>Dependency detection</td>
<td>requires static analysis or runtime instrumentation</td>
<td>directly supported</td>
</tr>
<tr>
<td>Control structures</td>
<td>rich set of control structures</td>
<td>Multiple workflow patterns</td>
</tr>
<tr>
<td>Evaluation of expressions</td>
<td>supported by a standard library</td>
<td>external operations or specific modules</td>
</tr>
<tr>
<td>Data conversions</td>
<td>supported by a standard library</td>
<td>require external operations</td>
</tr>
</tbody>
</table>
## Summary (2)

<table>
<thead>
<tr>
<th>Features</th>
<th>Scripts</th>
<th>Workflows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive execution</td>
<td>interactive interpreter</td>
<td>dynamic execution and refinement</td>
</tr>
<tr>
<td>Scheduling</td>
<td>requires dependency detection</td>
<td>dependency-based</td>
</tr>
<tr>
<td>Formal model</td>
<td>difficult to reason about program execution</td>
<td>various reasoning and refinement techniques</td>
</tr>
<tr>
<td>Checkpointing</td>
<td>interpreter state is difficult to save</td>
<td>can be saved persistently</td>
</tr>
<tr>
<td>Automatic composition</td>
<td>advanced code assist in development tools</td>
<td>possible to automatically construct abstract workflows</td>
</tr>
<tr>
<td>Execution engine</td>
<td>based on a standard interpreter</td>
<td>need a specialized execution engine</td>
</tr>
</tbody>
</table>
Thanks to

– Runtime system
  • Tomasz Gubala, Marek Kasztelnik, Piotr Nowakowski, Eryk Ciepiela, Asia Kocot
– Middleware
  • Maciej Malawski, Tomasz Bartynski, Jan Meizner
– Presentation and collaboration tools
  • Wlodzimierz Funika, Dariusz Krol, Daniel Harezlak, Alfredo Tirado
– Data Access
  • Matthias Assel, Stefan Wesner, (Aenne Loehden, Bettina Krammer), Piotr Nowakowski
– Provenance
  • Bartosz Balis, Jakub Wach, Michal Pelczar
– Integration
  • Tomasz Gubala, Marek Kasztelnik

• ViroLab Project (Peter Sloot)  
www.virolab.org
• VLvl description, demos, downloads  
virolab.cyfronet.pl