Interoperable Workflow Systems for Scientific Applications

Thomas Fahringer
Institute of Computer Science
University of Innsbruck
Austria

ISI, USC, Los Angeles
July 12, 2013
Distributed and Parallel Systems, UIBK

Research Interests

- **Cloud Computing**
  - Hardware and Software Virtualisation
  - Performance Modelling and Analysis
  - Quality of Service
  - Multi-criteria Scheduling
  - Service Level Agreements

- **Grid Computing**
  - Programming Paradigms and Methods
  - Meta Scheduling
  - Resource Brokerage
  - Performance Measurement, Analysis and Prediction
  - Ontologies for the Grid

- **Parallel Processing**
  - Programming Paradigms
  - Performance Instrumentation and Measurement
  - Performance Analysis and Interpretation
  - Performance Prediction
  - Computer Analysis and Optimisation
  - Experiment Management
ASKALON Grid/Cloud Computing Environment

http://www.askalon.org

Workflow Composition

AGWL

Execution Engine

Scheduler

Resource Manager

WSRF

Data Repository

Performance Analysis

GroudSim

Globus

Eucalyptus

Amazon EC2
ASKALON Workflow Development Stack

Invisible Cloud / Grid

AGWL
Abstract Grid Workflow Language

Visible Cloud / Grid

CGWR
Concrete Grid Workflow Representation

Portal

UML Workflow UML model
XML Activity Type
Java Activity Type
ASKALON Activity Deployment
Cloud / Grid Activity Instance

Application Developer

ASKALON Middleware

Cloud / Grid Infrastructure

Concretizing
Background

• **E-Science**
  → faster, better or innovative research
  → more comprehensive and larger problems
  → better results in shorter time
  → as simple as possible without losing performance

• **Scientific Workflows**
  → describe experiments conducted by scientists
  → tasks often legacy codes shared among scientists
  → target architectures: Grids, Clouds, HPC, desktop systems, etc.

• **Scientific Workflow Management Systems (WF systems)**
  → design and implement
  → execute and optimize
  → monitor and analyze scientific workflow applications
Complex WF Evolution
Interoperability Problem

• Most WF Systems have their own community
  → lock-in
  → sustainability

• Wouldn’t it be nice ...
  → develop WF in one WF system and reuse for others
  → run different parts of a WF with different WF systems
  → share meta-data, static and dynamic WF information (provenance)
  → reuse specific WF services

• Quick solutions for interoperability problems
  → use my system
  → use BPEL
  → XPDL (extensible process definition language)

• Our contribution: EU funded SHIWA project
The Project

Contract n°: 261585
Project type: I3
Start date: 01/07/2010
Duration: 24 months
Total budget: 2,101,980 €
Funding from the EC: 1,800,000 €
Total funded effort in person-months: 231
Web site: www.shiwa-workflow.eu
Contact person: Prof. Peter Kacsuk, email: Kacsuk@sztaki.hu
Project Partners

- MTA SZTAKI, Hungary
- UIBK, Austria
- C-UB, Germany
- CNRS, France
- UOW, United Kingdom
- CU, United Kingdom
- AMC, Netherlands
Project goals

• To leverage existing workflow based solutions
  → enable cross workflow exploitation of DCIs
  → applying both coarse- and fine-grained strategies

• Coarse-grained (CG) approach
  → combine existing workflow applications written in various workflow languages
  → treats existing workflows as black box systems
  → can be incorporated into other workflow applications as workflow nodes

• Language interoperability by defining an intermediate representation (FG)
  → used for translation of workflows across various systems
    ASKALON, P-Grade, MOTEUR, Triana, (Pegasus)

• SHIWA Simulation Platform
• SHIWA Repository
• SHIWA Portal
Shiwa Repository
Publish executable workflows in the Shiwa repository

SHIWA Repository facilitates **publishing** and **sharing** workflows

**Supports:**
- Abstract workflows with multiple implementations of multiple workflow systems
- Storing execution specific data

**Available:**
- accessible from the SHIWA Portal
Run workflows with your own data

- **SHIWA Repository**: Analyze description, inputs and outputs of published WFs
- **SHIWA Portal**: Instantiate WF from repo, execute with given sample data (inside WS-PGRADE workflow used as the Master WF system)
Coarse- / Fine-Grained Interoperability (CGI / FGI)

- CGI: nested / meta-workflows
- FGI: common workflow language (IWIR)

```java
Foreach d in D {
    A(d);
}
```
Foreach d in D {
    A(d);
}

Workflow language

Data transfers

Distributed data

Invocations

DCI 1

DCI 2

Master workflow system

Embedded workflow system

Front-end

Core engine

Back-end

Application services

Computing resources

Application services

Computing resources

CGI
Create and execute meta-workflows

Coarse-grained interoperability (CGI) = Nesting of different workflow systems to achieve interoperability of WF execution frameworks

SHIWA Portal

Meta-workflow

SHIWA Repository

WF System A

WF System B

WF System C
Foreach d in D {
    A(d);
}
SHIWA Desktop for FGI

Native DCI of A

Native DCI of B

Native DCI of C

WF A

WF B

WF C

SHIWA Repository

FGI Bundle of WF B

Cloud OpenStack

WF C containing WF B

WF C containing WF B

New DCI for C

IWIR transformation

WF B

WF B

WF B

WF B

WF B
Transform one WF to another WF system
Fine-grained interoperability (FGI)

Interoperable Workflow Intermediate Representation (IWIR)

Export to SHIWA repo as FGI bundle (IWIR+JSDL)

Import from SHIWA repo as FGI bundle (IWIR+JSDL)

WF_A

WF_B
Perform parallel processing on multiple DCIs

**DCI Interoperability**
SHIWA Bundles

- A SHIWA bundle is an object (stored as a zip file) containing everything needed to expose a workflow for use.
  - workflows, metadata, data and infrastructure dependencies, and other resources, such as multiple media types including text, images, data, and video.
  - A web “aggregation” (Object Reuse Exchange OAI standard)

- Provides a common language/format for workflow engines
Fine-Grain Interoperability Idea

- Workflow language interoperability
- DCI backend interoperability
Interoperable Workflow Intermediate Representation

- IWIR XML-based language

- Targets the **abstract part** of workflows
  - Concrete tasks handled through additional technologies

- Building Blocks:
  - Tasks
    - Atomic Tasks
    - Compound Tasks
  - Data/Control-Flow Links
IWIR Data Types

• Primitive types
  – boolean, double, integer, string, file

• Collection types
  – Ordered, indexed list of elements of the same data type

```
<type>::= <simple-type> | <collection-type>
<collection-type>::= "collection/"<type>
<simple-type>::= "string" | "integer" | "double" | "file" | "boolean"
```
Atomic Task

• Atomic Tasks define the signature of a computational task
  – Input port names and data-types
  – Output port names and data-types
  – Task type

• This is the binding element to the concrete part of the workflow
Compound Tasks

• Basic / Sequential compound tasks
  – If, BlockScope
  – While, For, ForEach

• Parallel compound tasks
  – parallelFor, parallelForEach

• Compound tasks completely encapsulate tasks and their data and control flow links
  – Workflows are tasks, and tasks are workflows
  – Enables easy reuse of sub-workflows
Data Flow in Compound Tasks

Sequential Loops

Parallel Loops

The if task
Examples

- Iteration strategies are defined between two or more collection-type input ports

- Dot product iteration
  - Execute a loop body once for each common collection index

- Cross product iteration
  - Execute a loop body once for every possible combination of collection elements
<parallelForEach name="forEach1">
  <inputPorts>
    <loopElements>
      <loopElement name="collA" type="collection/file" />
      <loopElement name="collB" type="collection/file" />
    </loopElements>
  </inputPorts>
  <body>
    <task name="A" tasktype="consumer">
      <inputPorts>
        <inputPort name="elementA" type="file" />
        <inputPort name="elementB" type="file" />
      </inputPorts>
      <outputPorts>
        <outputPort name="res" type="file" />
      </outputPorts>
    </task>
  </body>
  <outputPorts>
    <outputPort name="res" type="collection/file" />
  </outputPorts>
  <links>
    <link from="#forEach1/collA" to="#A/elementA" />
    <link from="#forEach1/collB" to="#A/elementB" />
    <link from="#A/res" to="#forEach1/res" />
  </links>
</parallelForEach>
Cross Product
WF Transformation from Moteur to Askalon

- **Image registration workflow**
  - Common medical image spatial alignment procedure

- **First**
  - Extract first image from the set

- **Register_to_first**
  - Align all images to first in the set

- **Average**
  - Compute the mean image

- **Register_to_average**
  - Align all resulting images to an average model to avoid any bias related to using the first image
GWENDIA to IWIR

```xml
<task name="First" tasktype="First">
  <inputPorts>
    <inputPort name="in" type="collection/file"/>
    <inputPort name="ref" type="file"/>
    <inputPort name="float" type="file"/>
  </inputPorts>
  <outputPorts>
    <outputPort name="out" type="file"/>
  </outputPorts>
</task>

<parallelForEach name="Register_to_first:cross">
  <inputPorts>
    <inputPort name="ref" type="file"/>
    <inputPort name="float" type="file"/>
  </inputPorts>
  <body>
    <task name="Register_to_first" tasktype="Register_to_first">
      <inputPorts>
        <inputPort name="ref" type="file"/>
        <inputPort name="float" type="file"/>
      </inputPorts>
      <outputPorts>
        <outputPort name="out" type="collection/file"/>
      </outputPorts>
      <links>
        <link from="Register_to_first:cross/ref" to="Register_to_first/ref"/>
        <link from="Register_to_first:cross/float" to="Register_to_first/float"/>
        <link from="Register_to_first/out" to="Register_to_first:cross/out"/>
      </links>
    </task>
  </body>
</parallelForEach>
```
Loading IWIR in ASKALON
IWIR Tool

- Parse IWIR XML files and provide a Java representation
- Construct IWIR workflows using a Java API
- Export workflows to IWIR-compliant XML files
- Traverse and manipulate IWIR workflows
- Parse and evaluate the condition expressions
- Validate IWIR documents against the XML Schema, data types, control flow and data flow

**IWIR tool & IWIR specification** available at:
Conclusions

- Fine-grain WF system interoperability
  - translate and share WF among WF systems
  - tool support for IWIR transformations and analysis

- Coarse-grain interoperability
  - meta-workflow that invokes various other enactment engines for sub-workflows
  - DCI bridge to different DCIs

- Back-end interoperability through Shiwa bundles

- Supported initially by the four important European workflow systems in the SHIWA project
  - ASKALON, MOTEUR, (WS-)P-GRADE, Triana

- IWIR specification and tools
  - http://www.shiwa-workflow.eu
Thanks!

http://www.shiwa-workflow.eu
Thank you.