Scientific Workflow Development Using both Visual and Script-based Representation

KETAN MAHESHWARI, JOHAN MONTAGNAT

Modalis Team, UNS / CNRS-I3S LABORATORY
Sophia Antipolis, France
http://modalis.polytech.unice.fr/

MODALIS

Overview

- A semantics preservation between a visual and a script-based workflow development.
- A script language, gscript is developed whose execution semantics matches the Gwendia language and its GUI-based workflow enactor–MOTEUR2.
- A validated EBNF grammar through the ANTLR language tool.
- Two-way translators to convert a source workflow into its semantically equivalent counter-part, using a single enactor.

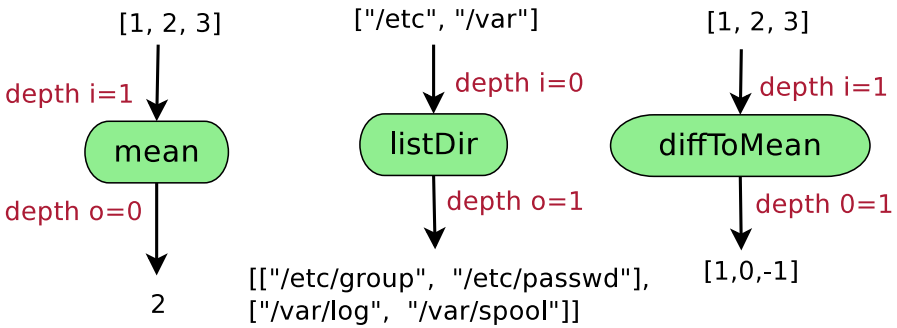
Motivation for a Script-based Workflow Representation

- Catering a broader user-base: novices and experts.
- Compact workflow representation supporting *parallelism implicitly*.
- Rapid workflow composition in portable form.
- Leveraging an existing workflow engine (MOTEUR2).

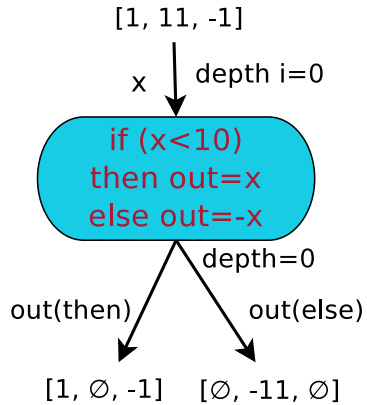
Similar work in the Scientific Workflow Community

SWE/Language	Composition	DCI Interfacing
Swift/SwiftScript	Script	Clusters
GridNexus/GXPL	GUI+script	Grids
OSyRIS/SiLK	GUI+script	WS-based
Wool/-	script	Independent
Vizbuilder/bioflow	GUI+script	Independent
Martlet/-	Script	multi-middleware
StarFlow/-	Script	Ext. Schedulers
M2/Gwendia/gscript	GUI+script	EGI, G5K

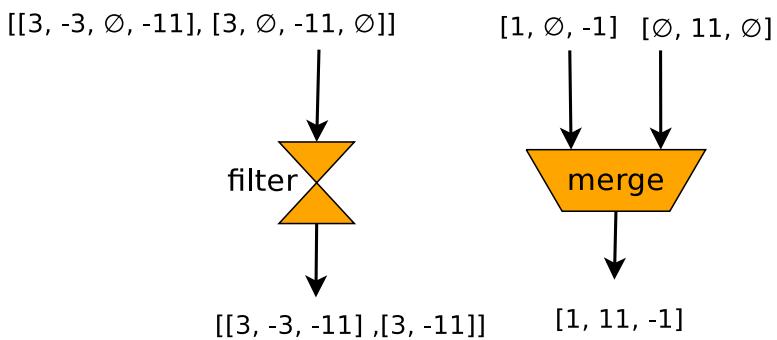
GWENDIA Language Highlights



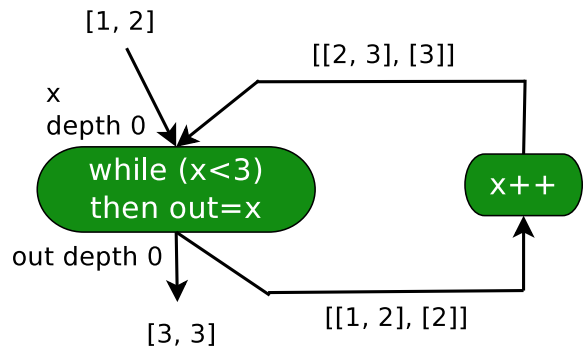
Array Programming Semantics



Conditional



Filter and Merge



Loop

[Details in “A Data-driven Workflow Language for Grids Based on Array Programming Principles”, Montagnat et. al., WORKS’09, Portland Oregon, USA]

Gscript is Semantically Same, Syntactically Different

gscript offers a script-based, non-verbose, functional programming style expression of GWENDIA workflows.

```
in1=["a","b","c"]

in2=[1,2,3]

p1out@0 = p1 (ws:http://aservice:op, cross(in1@0,in2@0))

wfout@0 = p2 (bs:"print(\" + p1out + \");", dot(p1out@0))
```

Anatomy of a gscript Data Declaration

```
in =  100
```

Scalar Declaration

```
in1 =  [ "a", "b", "c" ]

in2 =  [ 1, 2, 3 ]
```

Array Declaration

Anatomy of a gscript Statement

p1out @ 0 = p1 (ws: http://service, cross (in1@0,in2@0))

- p1out → outport
- @ 0 → portdepth
- p1 → processor
- ws: → invoker (service)
- http://service → service URI
- cross → iterator
- in1, in2 → inports

MODALIS

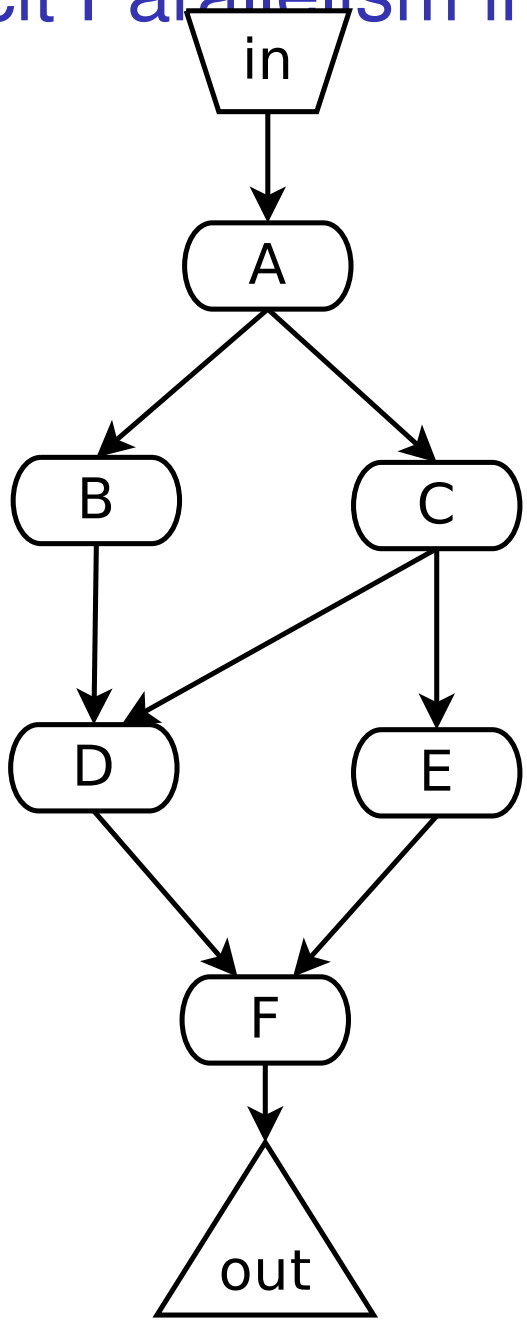
Use of **Futures** in gscsript

- The use of *one-time assignment* future variables in gscript makes code implicitly parallel.
- `a=f(x); b=g(y); c=h(a)`, assignment of a and b are non-blocking and f() and g() are executed concurrently while assignment of c is blocking until a is computed.
- The execution profile of the resulting workflow is asynchronous and data-flow driven.
- The result is a highly expressive parallel workflows driven by the best-effort parallel execution.

Example Follows ...

MODALIS

Explicit Parallelism in Parallel Languages

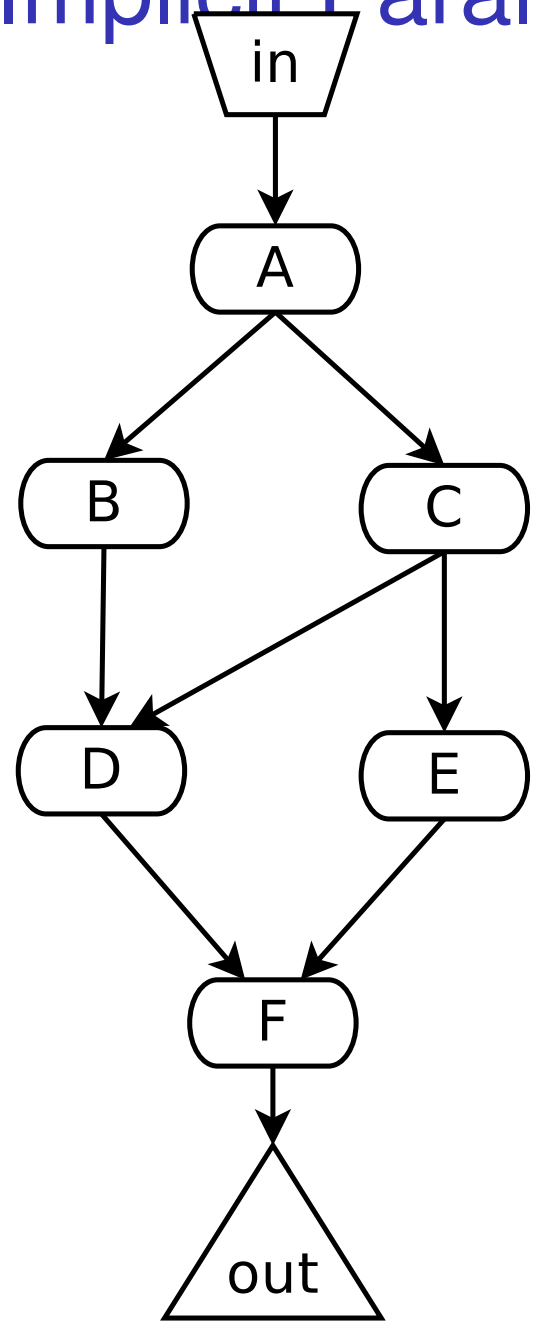


Proc B and E are invoked sequentially **despite** being *independent*.

```
exec A
dopar {
exec B, exec C
}
dopar {
exec D, exec E
}
exec F
```

MODALIS

Implicit Parallelism in Gscript

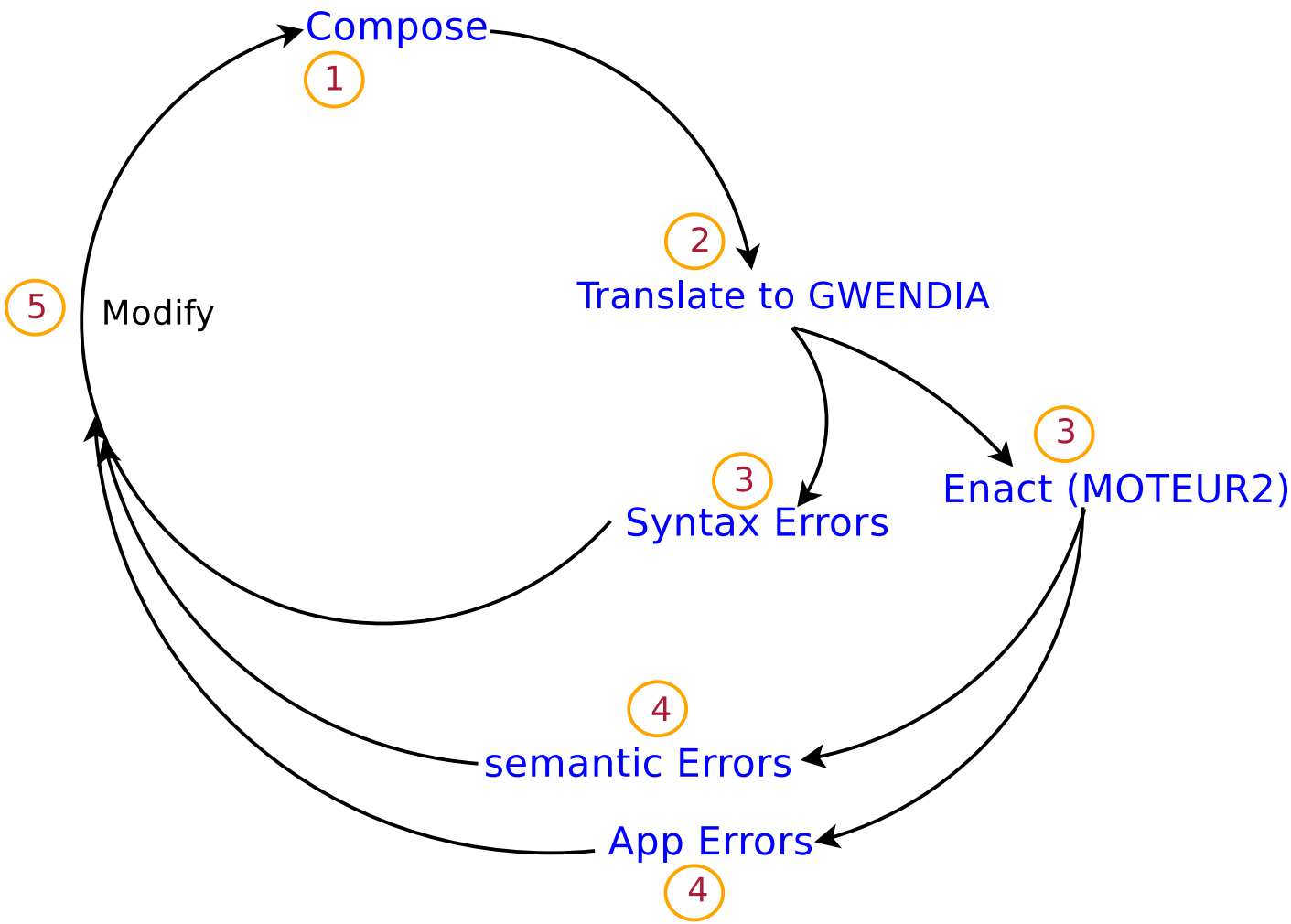


All proc are invoked simultaneously; (future) variables are evaluated based on their **requirement.**

```
aout@0 = A (<invoker>,in@0)
bout@0 = B (<invoker>,aout@0)
cout@0 = C (<invoker>,aout@0)
dout@0 = D (<invoker>,dot(bout@0,cout@0))
eout@0 = E (<invoker>,cout@0)
wfout@0 = F (<invoker>,dot(dout@0,eout@0))
```

MODALIS

The gscript Workflow Development Cycle



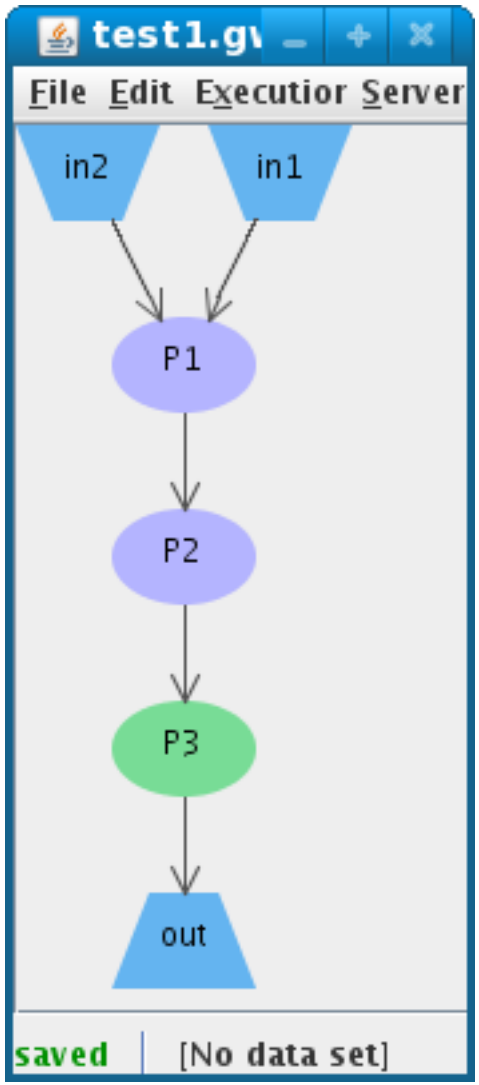
GWENDIA $\longleftrightarrow$ gscript Translation

- **Phase 1:** Data specification
- **Phase 2:** Processors, ports, depth, iterators
- **Phase 3:** Links, write to file

Implementation

- Data specification translator
- gs2gwendia, gwendia2gs within the MOTEUR2 package

Example: Visual vs. Text




```
# input data starts
in2=[in2_1,in2_2]
in1=[in1_1,in1_2]
# input data ends

P1_out@0=P1(bs:"print␣\"hello Proc1\";",
           cross(in1_out2@0,in1_out@0))

P3_out@0=P3(ws:"http://service:action",dot(P2_out@0))

P2_out@0=P2(bs:"print␣\"hello P2\";",dot(P1_out@0))
```

... vs. XML

```
  <?xml version="1.0" encoding="UTF-8"?>
<workflow name="test1.gwendia">

  <interface>
    <source name="in1" type="string" />
    <source name="in2" type="string" />
    <sink name="out" type="string" />
  </interface>

  <processors>
    <processor name="P1" >
      <in name="in2" type="string" depth="0" />
      <in name="in" type="string" depth="0" />
      <out name="out" type="string" depth="0" />
      <iterationstrategy>
        <cross>
          <port name="in" />
          <port name="in2" />
        </cross>
      </iterationstrategy>
      <beanshell>print "hello␣Proc1";
      </beanshell>
    </processor>
```

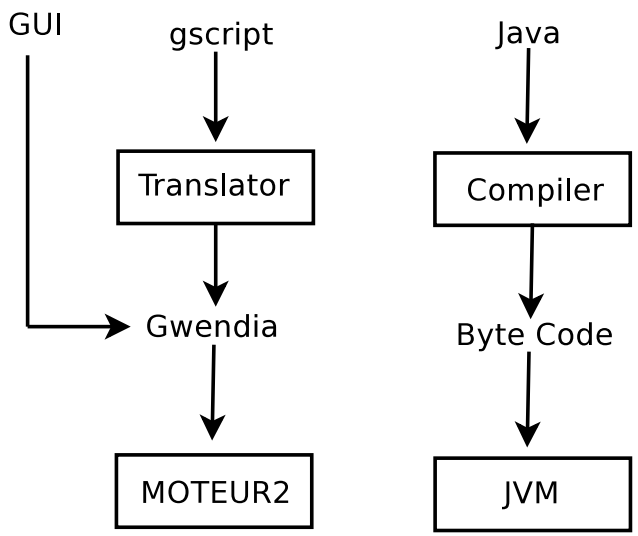
contd.

```
 <processor name="P3" >
      <in name="in" type="string" depth="0" />
      <out name="out" type="string" depth="0" />
      <webservice wsdl="http://service" operation="null" />
    </processor>
    <processor name="P2" >
      <in name="in" type="string" depth="0" />
      <out name="out" type="string" depth="0" />
      <beanshell>print "hello␣P2";
      </beanshell>
    </processor>
  </processors>

  <links>
      <link from="in1" to="P1:in" />
      <link from="in2" to="P1:in2" />
      <link from="P1:out" to="P2:in" />
      <link from="P2:out" to="P3:in" />
      <link from="P3:out" to="out" />
  </links>

</workflow>
```

Related Work: Swiftscript/karajan

- Similar approach: script $\longrightarrow$ xml $\longrightarrow$ workflow enactor
- Different in array semantics, especially multi-dimension arrays: no array-portdepth semantics
- No direct support for different iteration strategies: can be achieved with other language constructs
- Swift has richer support for diverse filesystems, gscript limits it to predefined invokers

Current Work: Proof for Semantics Preservation



- Similar to the programming languages scenario.
- A high-level language is compiled into a low-level language before being executed.

Semantics Preservation Requirements

- What we have:
  gscript grammar, Gwendia schema
- What we need:
  To show that the two grammar structures are isomorphic

Conclusions

- Two-way workflow composition using single engine
- Rapid workflow prototyping
- Comparable with swiftscript in terms of compactness of workflow expression

Thank You! Questions?

Acknowledgment:
This work is supported by the French ANR GWENDIA project under contract number ANR-06-MDCA-009.
http://gwendia.polytech.unice.fr

MODALIS