Domain-Specific Languages for Composing Signature Discovery Workflows

Ferosh Jacob*, Adam Wynne+, Yan Liu+, Nathan Baker+, and Jeff Gray*

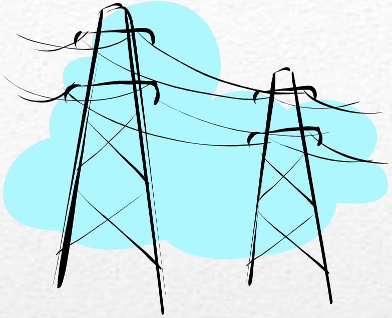
*Department of Computer Science, University of Alabama, AL

+Pacific Northwest National Laboratory, Richland, WA

THE UNIVERSITY OF
ALABAMA
COMPUTER SCIENCE

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Signature Discovery Initiative (SDI)



The most widely understood signature is the human fingerprint

Biomarkers can be used to indicate the presence of disease or identify a drug resistance

A signature is a unique or distinguishing measurement, pattern or collection of data that identifies a phenomenon (object, action or behavior) of interest

Combinations of line overloads that may lead to a cascading power failure

Anomalous network traffic is often an indicator of a computer virus or malware

- **Anticipate** future events by detecting precursor signatures, such as combinations of line overloads that may lead to a cascading power failure
- **Characterize** current conditions by matching observations against known signatures, such as the characterization of chemical processes via comparisons against known emission spectra
- **Analyze** past events by examining signatures left behind, such as the identity of cyber hackers whose techniques conform to known strategies and patterns

SDI high-level goals

SDI Analytic Framework (AF)

Challenge:
An approach that can be applied **across a broad spectrum** to efficiently and robustly construct candidate signatures, validate their reliability, measure their quality and overcome the challenge of detection

Solution: Analytic Framework (AF)

- Legacy code in a remote machine is wrapped and exposed as web services,
- Web services are orchestrated to create re-usable tasks that can be retrieved and executed by users

- Accidental complexity of creating service wrappers
  - ❖ In our system, manually wrapping a simple script that has a single input and output file requires 121 lines of Java code (in five Java classes) and 35 lines of XML code (in two files).
- Lack of end-user environment support
  - ❖ Many scientists are not familiar with service-oriented software technologies, which force them to seek the help of software developers to make Web services available in a workflow workbench.

Challenges for scientists in using AF

We applied Domain-Specific Modeling (DSM) techniques to

- **Model** the process of wrapping remote executables.

  The executables are wrapped inside AF web services using a Domain-Specific Language (DSL) called the Service Description Language (SDL).

- **Model** the SDL-created web services

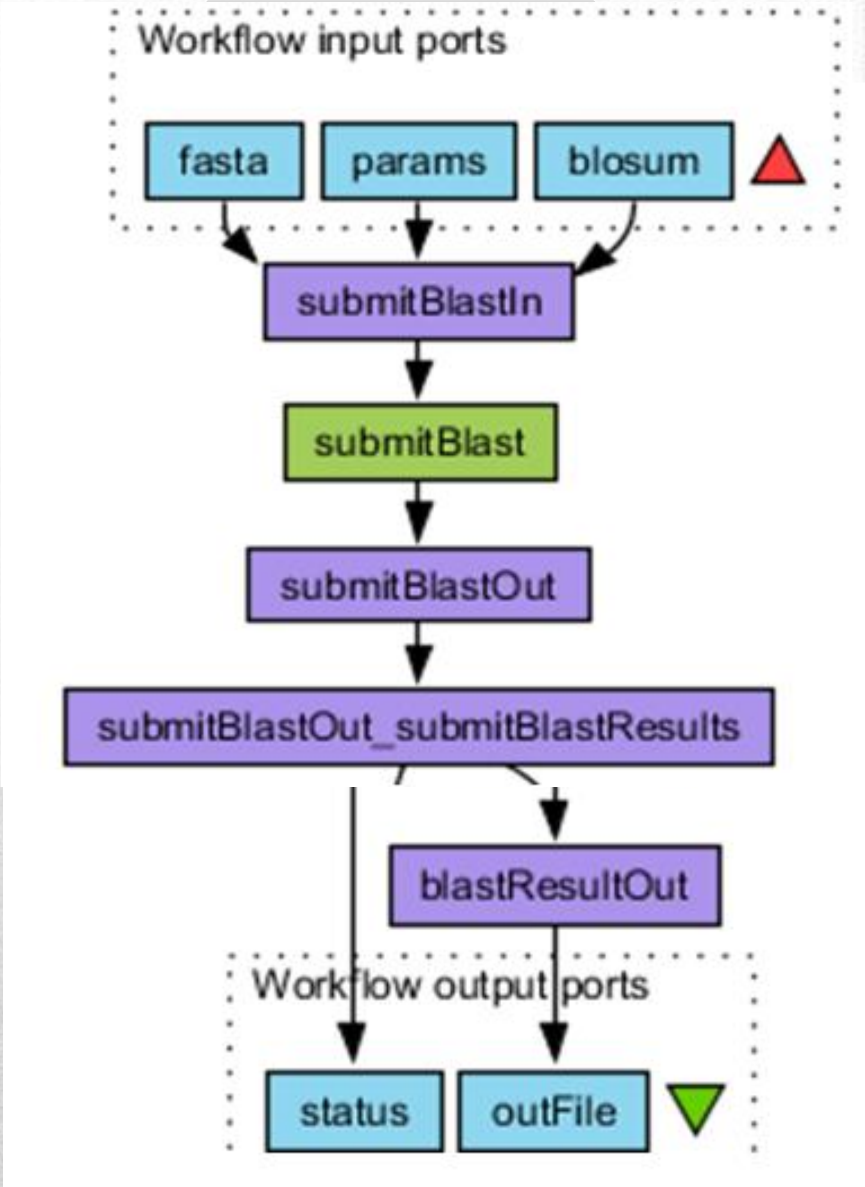
  The SDL-created web services can then be used to compose workflows using another DSL, called the Workflow Description Language (WDL).

A domain-specific modeling approach

1. Submit job

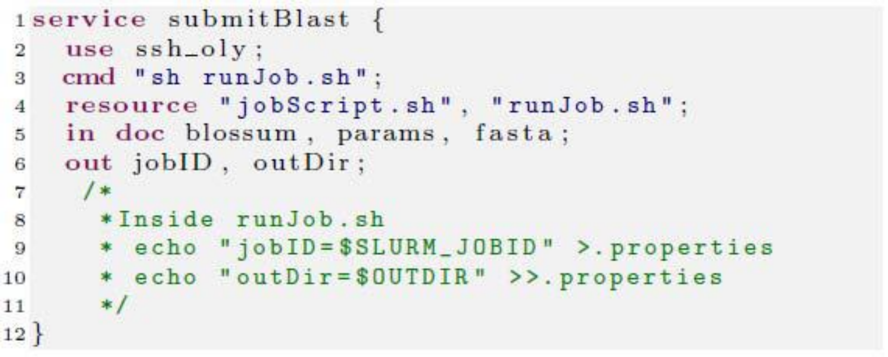
2. Check status

3. Download files



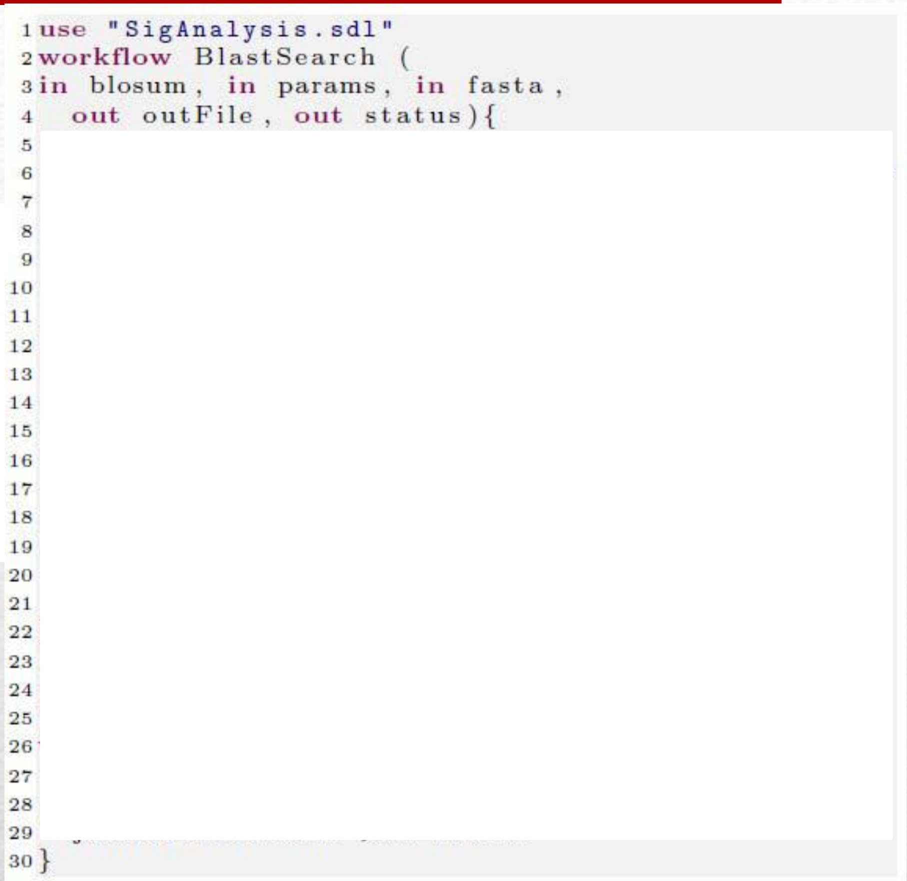
Output generated as Taverna workflow executable

Example application: BLAST execution



```
service submitBlast {
  use ssh_oly;
  cmd "sh runJob.sh";
  resource "jobScript.sh", "runJob.sh";
  in doc blossum, params, fasta;
  out jobID, outDir;
   /*
    *Inside runJob.sh
    * echo "jobID=$SLURM_JOBID" >.properties
    * echo "outDir=$OUTDIR" >>.properties
    */
}
```

Service description (SDL) for BLAST submission



```
use "SigAnalysis.sdl"
workflow BlastSearch (
in blosum, in params, in fasta,
  out outFile, out status){
}
```

Workflow description (WDL) for BLAST

Implementation

Script metadata
(e.g., name, inputs)

SDL
(e.g., blast.sdl)

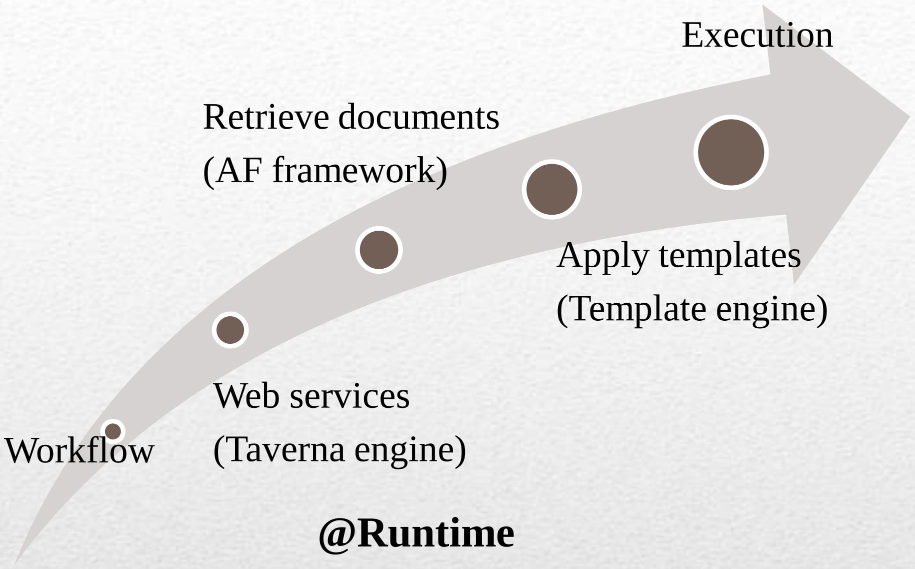
WDL
(e.g., blast.wdl)

Inputs

Web services
(e.g., checkJob)

Taverna workflow
(e.g., blast.t2flow)

Outputs



Workflow

Web services
(Taverna engine)

Retrieve documents
(AF framework)

Apply templates
(Template engine)

Execution

@Runtime

- Compared to domain-independent workflows like JBPM and Taverna, our framework has the advantage that it is configured only for scientific signature discovery workflows.
- Most of these tools assume that the web services are available. Our framework configures the workflow definition file that declares how to compose services wrappers created by our framework.

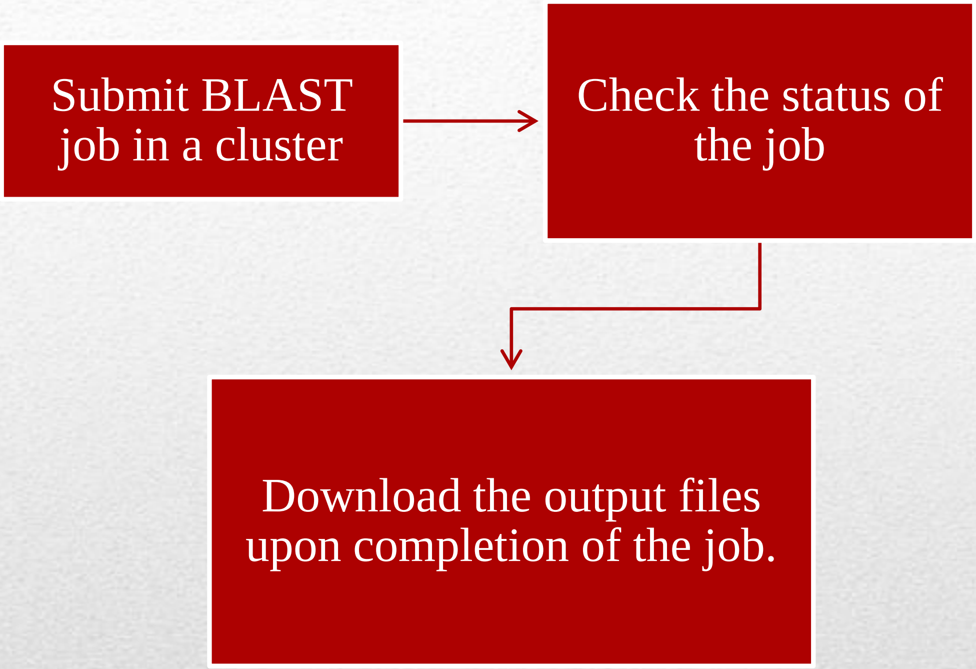
Related works

We successfully designed and implemented two DSLs (SDL and WDL) for converting remote executables into scientific workflows. SDL can generate services that are deployable in a signature discovery workflow using WDL. We separated the domain-specific information required to create the workflows from the accidental complexities introduced by webservices and the Taverna workflow engine, which allows end-users (scientists) to design and develop workflows

Summary

Questions ?

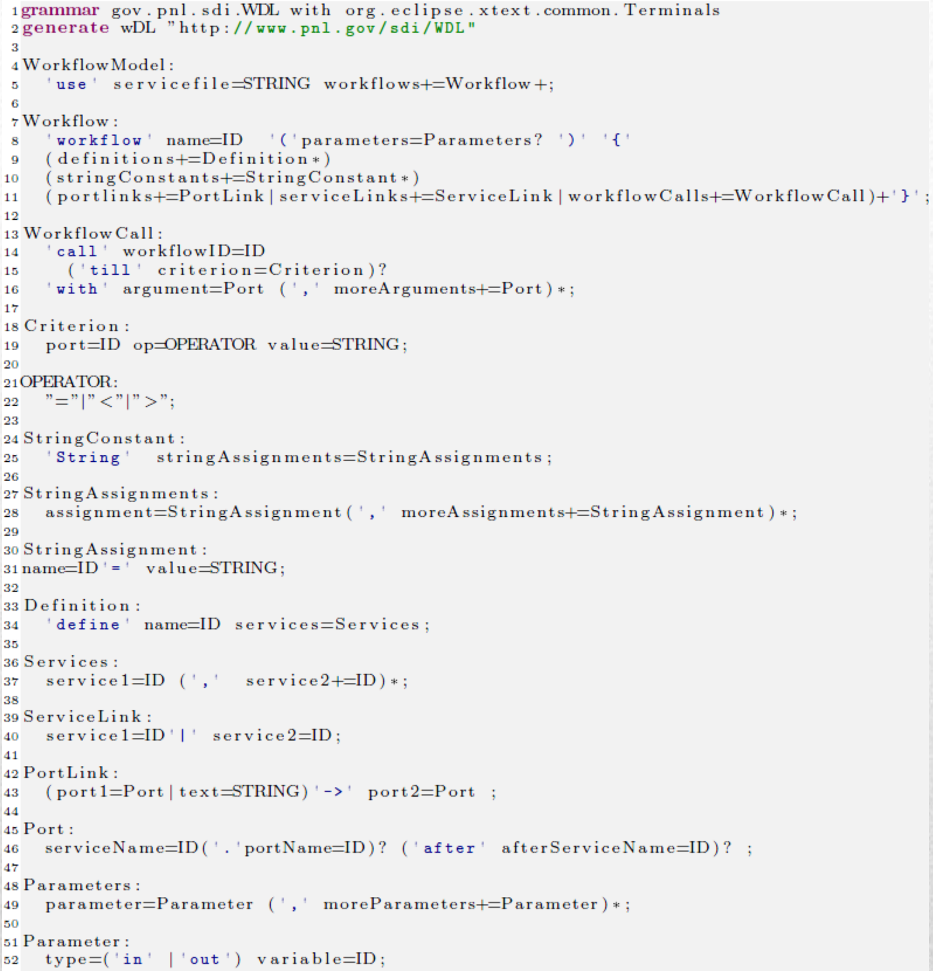
Example application: BLAST execution



Submit BLAST job in a cluster

Check the status of the job

Download the output files upon completion of the job.

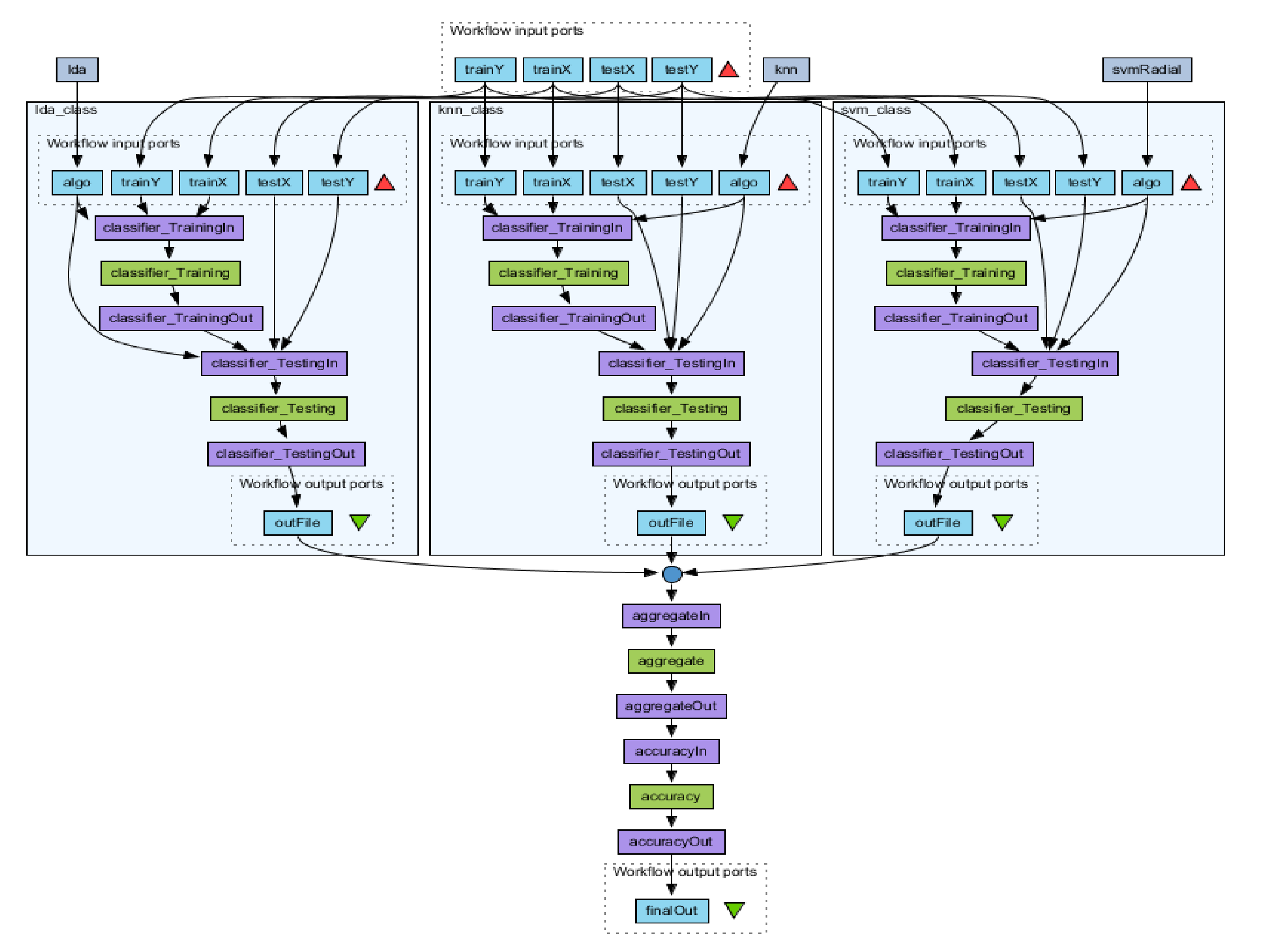


```
grammar gov.pnl.sdi.WDL with org.eclipse.xtext.common.Terminals
generate wDL "http://www.pnl.gov/sdi/WDL"

WorkflowModel:
  'use' servicefile=STRING workflows+=Workflow+;

Workflow:
  'workflow' name=ID  '('parameters=Parameters? ')' '{'
  (definitions+=Definition*)
  (stringConstants+=StringConstant*)
  (portlinks+=PortLink|serviceLinks+=ServiceLink|workflowCalls+=WorkflowCall)+'}';

WorkflowCall:
  'call' workflowID=ID
    ('till' criterion=Criterion)?
  'with' argument=Port (',' moreArguments+=Port)*;

Criterion:
  port=ID op=OPERATOR value=STRING;

OPERATOR:
  "="|"<"|">";

StringConstant:
  'String'  stringAssignments=StringAssignments;

StringAssignments:
  assignment=StringAssignment(',' moreAssignments+=StringAssignment)*;

StringAssignment:
name=ID'=' value=STRING;

Definition:
  'define' name=ID services=Services;

Services:
  service1=ID (','  service2+=ID)*;

ServiceLink:
  service1=ID'|' service2=ID;

PortLink:
  (port1=Port|text=STRING)'->' port2=Port ;

Port:
  serviceName=ID('.'portName=ID)? ('after' afterServiceName=ID)? ;

Parameters:
  parameter=Parameter (',' moreParameters+=Parameter)*;

Parameter:
  type=('in' |'out') variable=ID;
```

Xtext grammar for WDL

No	Service	Utils/Script	{Inputs (type)] [Outputs(type)]	LOC	Total LOC (files)
1	echoString	echo	[0][1 (doc)]	10+13+1+6	30(4)
2	echoFile	echo	[1 (String)] [1 (doc)]	10+14+1+6	31(4)
3	aggregate	cat	[1(List doc)] [1 (doc)]	10+20+1+7	38(4)
4	classifier_Training	R	[2 (doc), 1 (String)] [1 (doc)]	11+24+2+8	45(4)
5	classifier_Testing	R	[3 (doc), 1 (String)] [1 (doc)]	12+29+2+8	51(4)
6	accuracy	R	[1 (doc)] [1 (doc)]	11+19+1+6	37(4)
7	submitBlast	SLURM, sh	[3 (doc)] [2 (String)]	17+27+2+8+18	72(5)
8	jobStatus	SLURM, sh	[1 (String)] [1 (String)]	10+14+1+6	31(4)
9	blastResult	cp	[1 (String)] [1 (doc)]	10+14+1+6	31(4)
10	mafft	mafft	[1 (doc)] [1 (doc)]	10+18+1+6	35(4)

An overview of SDL code generation



Taverna classification workflow