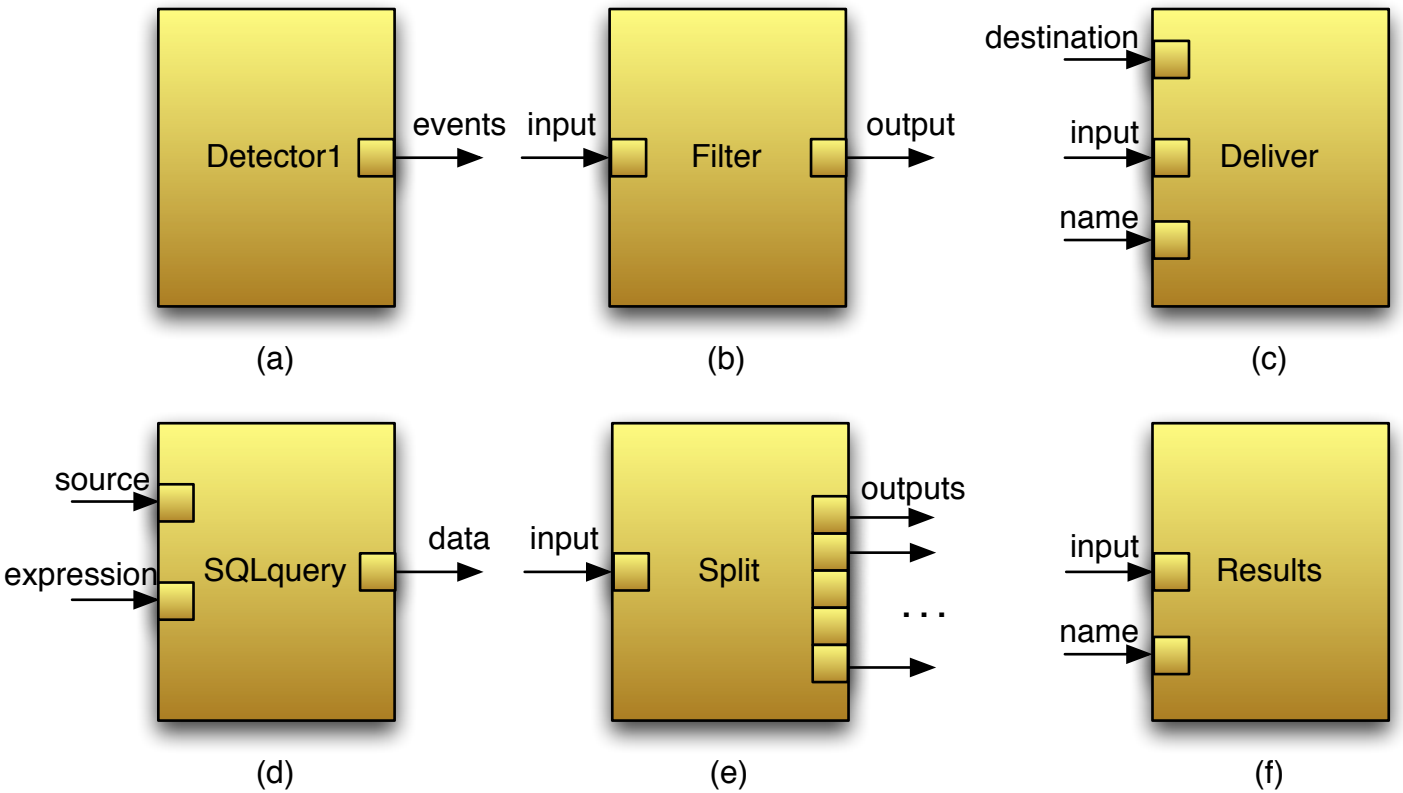
The Data-Intensive Systems Process Engineering Language — A Tutorial

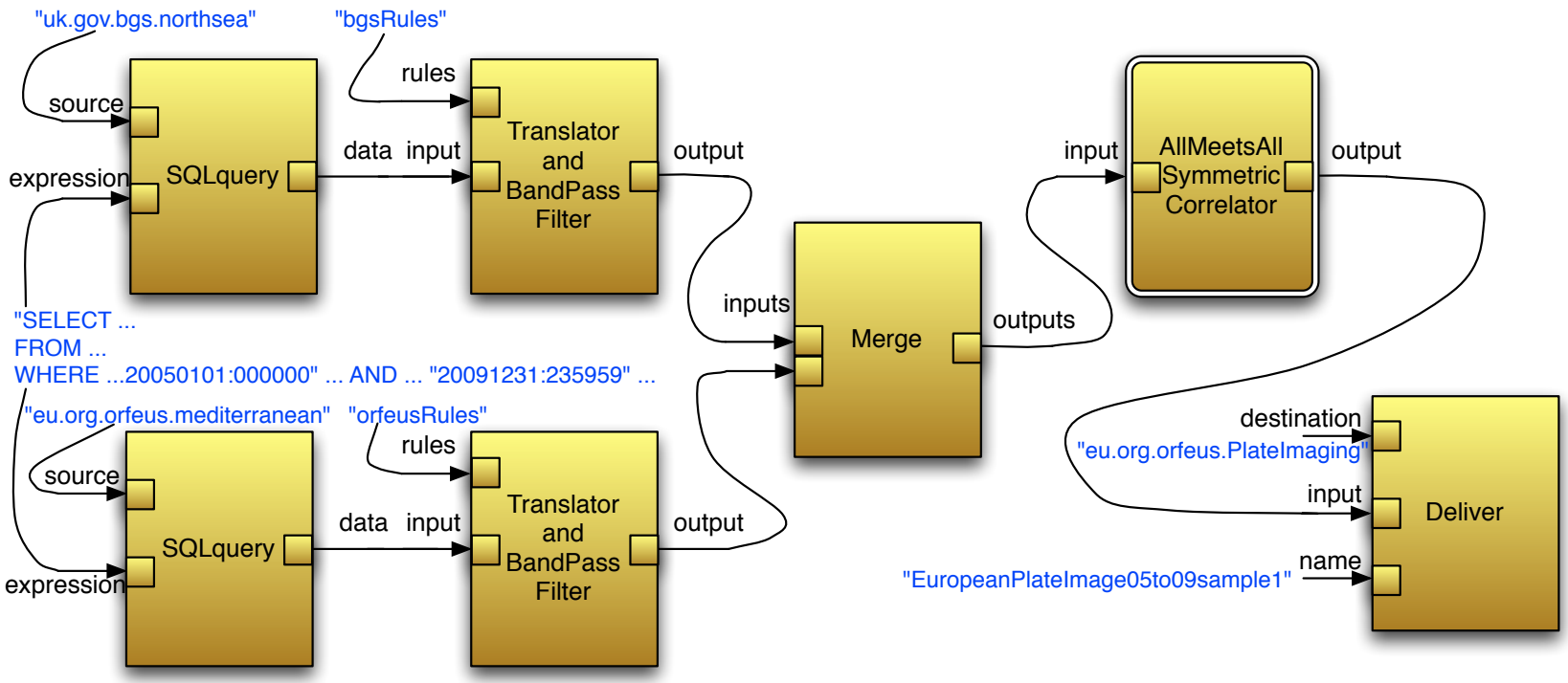
Paul Martin

DIR Group, 22nd of April 2011

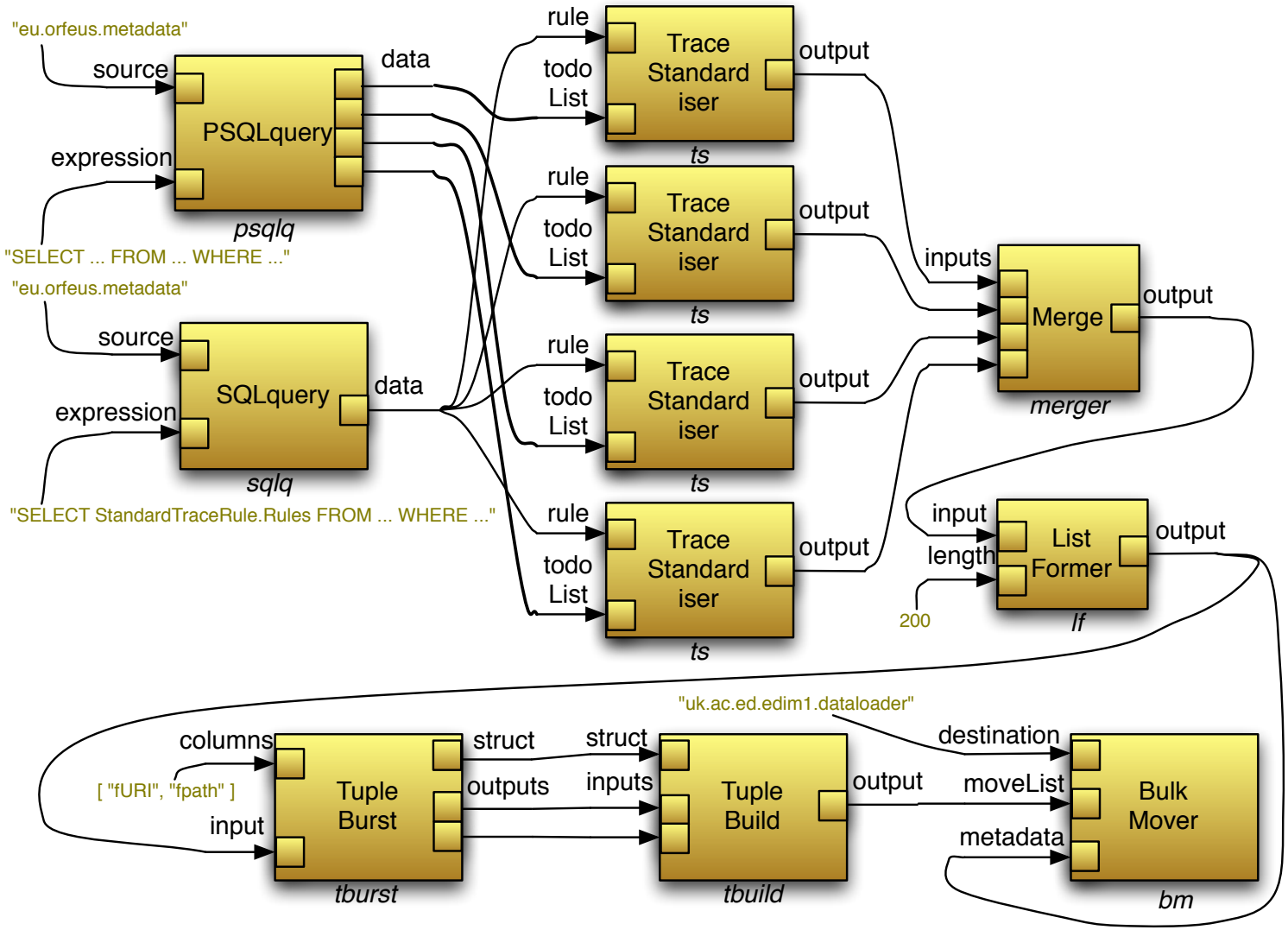
Processing Elements



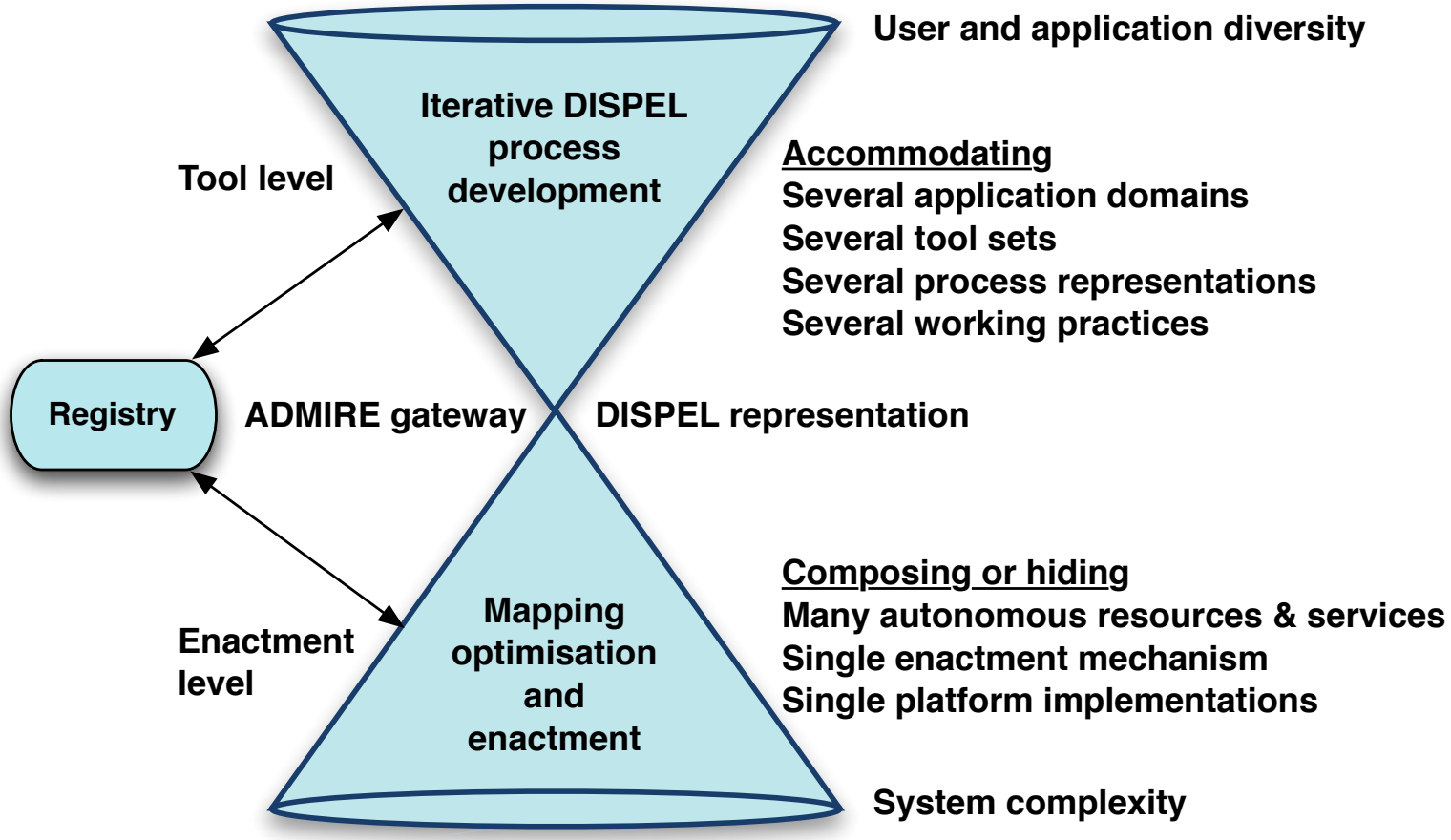
Workflows


"uk.gov.bgs.northsea"
"bgsRules"
source
rules
SQLquery
Translator and BandPass Filter
expression
data input
output
input
AllMeetsAll Symmetric Correlator
output
inputs
Merge
outputs
"SELECT ...
FROM ...
WHERE ...20050101:000000" ... AND ... "20091231:235959" ...
"eu.org.orfeus.mediterranean"
"orfeusRules"
source
rules
SQLquery
Translator and BandPass Filter
expression
data input
output
destination
"eu.org.orfeus.PlateImaging"
input
Deliver
name
"EuropeanPlateImage05to09sample1"

Workflows


"eu.orfeus.metadata"
source
expression
PSQLquery
data
psqlq
"SELECT ... FROM ... WHERE ..."
"eu.orfeus.metadata"
source
expression
SQLquery
data
sqlq
"SELECT StandardTraceRule.Rules FROM ... WHERE ..."
rule
todo List
Trace Standard iser
output
ts
rule
todo List
Trace Standard iser
output
ts
rule
todo List
Trace Standard iser
output
ts
rule
todo List
Trace Standard iser
output
ts
inputs
Merge
output
merger
input
length
List Former
output
lf
200
"uk.ac.ed.edim1.dataloader"
columns
["fURI", "fpath"]
input
Tuple Burst
struct
outputs
tburst
struct
inputs
Tuple Build
output
tbuild
destination
moveList
metadata
Bulk Mover
bm

ADMIRE Architecture



Submitting Workflows

```
1  // Import PEs from the registry.
2  use uk.org.ogsadai.SQLQuery;
3  use eu.admire.Recorder;
4
5  // Create instances of PEs.
6  SQLQuery sqlq     = new SQLQuery;
7  Recorder recorder = new Recorder;
8
9  // Connect PEIs together to create workflow.
10 |-"SELECT name FROM AtlanticSurveys"-| => sqlq.expression;
11                    |-"uk.org.UoE.dbA"-| => sqlq.resource;
12                                 sqlq.data => recorder.input;
13
14 // Submit the workflow (by submitting the final component).
15 submit recorder;
```

Using Streams in Workflows

```
1  // Import PEs from the registry.
2  use uk.org.ogsadai.SQLQuery;
3  use eu.admire.Recorder;
4
5  // Create instances of PEs.
6  SQLQuery sqlq     = new SQLQuery;
7  Recorder recorder = new Recorder;
8
9  // A batch of queries.
10 String query1 = "SELECT * FROM AtlanticSurveys WHERE" +
11                 " AtlanticSurveys.date before '2005' AND" +
12                 " AtlanticSurveys.date after '2000' AND" +
13                 " AtlanticSurveys.latitude >= 0";
14 String query2 = "SELECT * FROM PacificSurveys WHERE" +
15                 " PacificSurveys.date before '2004' AND" +
16                 " PacificSurveys.date after '2001'";
17 String query3 = "SELECT age FROM PacificSurveys";
18
19 // A list of resources.
20 String database1 = "uk.org.UoE.dbA";
21 String database2 = "uk.org.UoE.dbB";
22 String database3 = "uk.org.UoE.dbC";
23
24 // Connect PEIs together to create workflow.
25          |-query1, query2, query3-| => sqlq.expression;
26 |-database1, database2, database3-| => sqlq.resource;
27                           sqlq.data => recorder.input;
28
29 // Submit the workflow (by submitting the final component).
30 submit recorder;
```

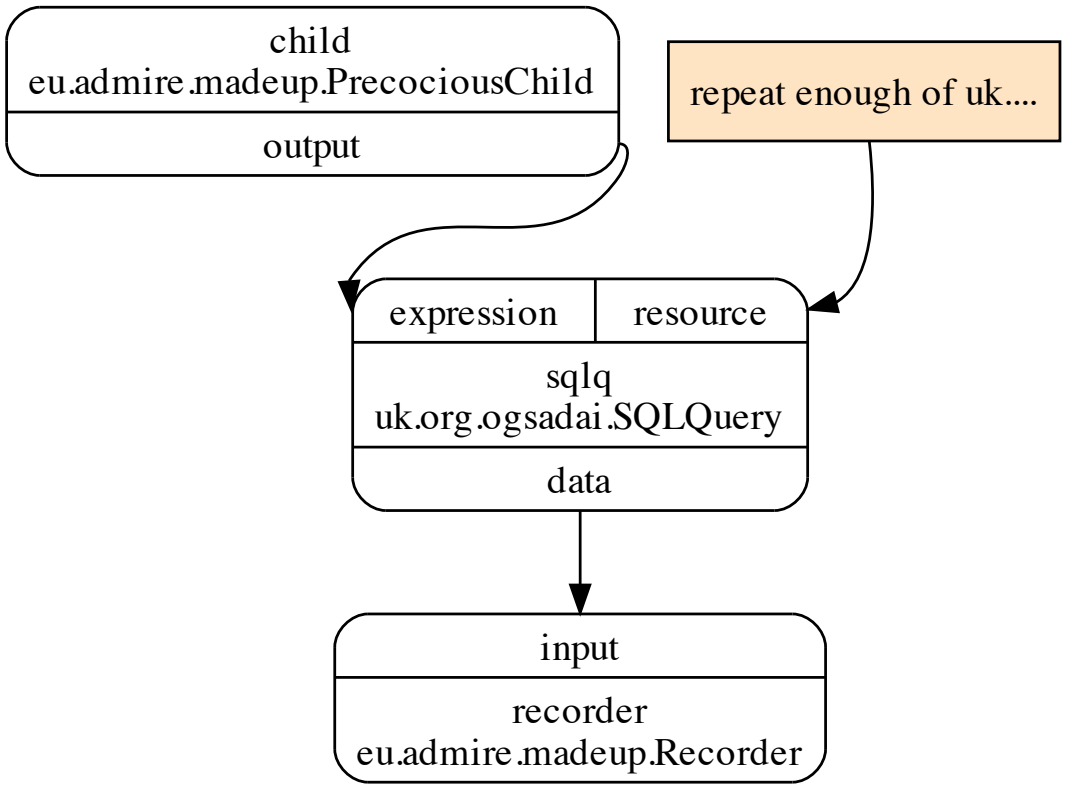
Using Streams in Workflows

```
1  // Import PEs from the registry.
2  use uk.org.ogsadai.SQLQuery;
3  use eu.admire.Recorder;
4
5  // Create instances of PEs.
6  SQLQuery sqlq     = new SQLQuery;
7  Recorder recorder = new Recorder;
8
9  // A batch of queries.
10 String query1 = "SELECT * FROM AtlanticSurveys WHERE" +
11                 " AtlanticSurveys.date before '2005' AND" +
12                 " AtlanticSurveys.date after '2000' AND" +
13                 " AtlanticSurveys.latitude >= 0";
14 String query2 = "SELECT * FROM PacificSurveys WHERE" +
15                 " PacificSurveys.date before '2004' AND" +
16                 " PacificSurveys.date after '2001'";
17 String query3 = "SELECT age FROM PacificSurveys";
18
19 // Connect PEIs together to create workflow.
20       |-query1, query2, query3-| => sqlq.expression;
21 |-repeat 3 of "uk.org.UoE.dbA"-| => sqlq.resource;
22                         sqlq.data => recorder.input;
23
24 // Submit the workflow (by submitting the final component).
25 submit recorder;
```

Using Streams in Workflows

```
1  // Import PEs from the registry.
2  use uk.org.ogsadai.SQLQuery;
3  use eu.admire.madeup.Recorder;
4  use eu.admire.madeup.PrecociousChild;
5
6  // Create instances of PEs.
7  PrecociousChild child    = new PrecociousChild;
8  SQLQuery        sqlq     = new SQLQuery;
9  Recorder        recorder = new Recorder;
10
11 // Connect PEIs together to create workflow.
12                            child.output => sqlq.expression;
13 |-repeat enough of "uk.org.UoE.dbA"-| => sqlq.resource;
14                               sqlq.data => recorder.input;
15
16 // Submit the workflow (by submitting the final component).
17 submit recorder;
```

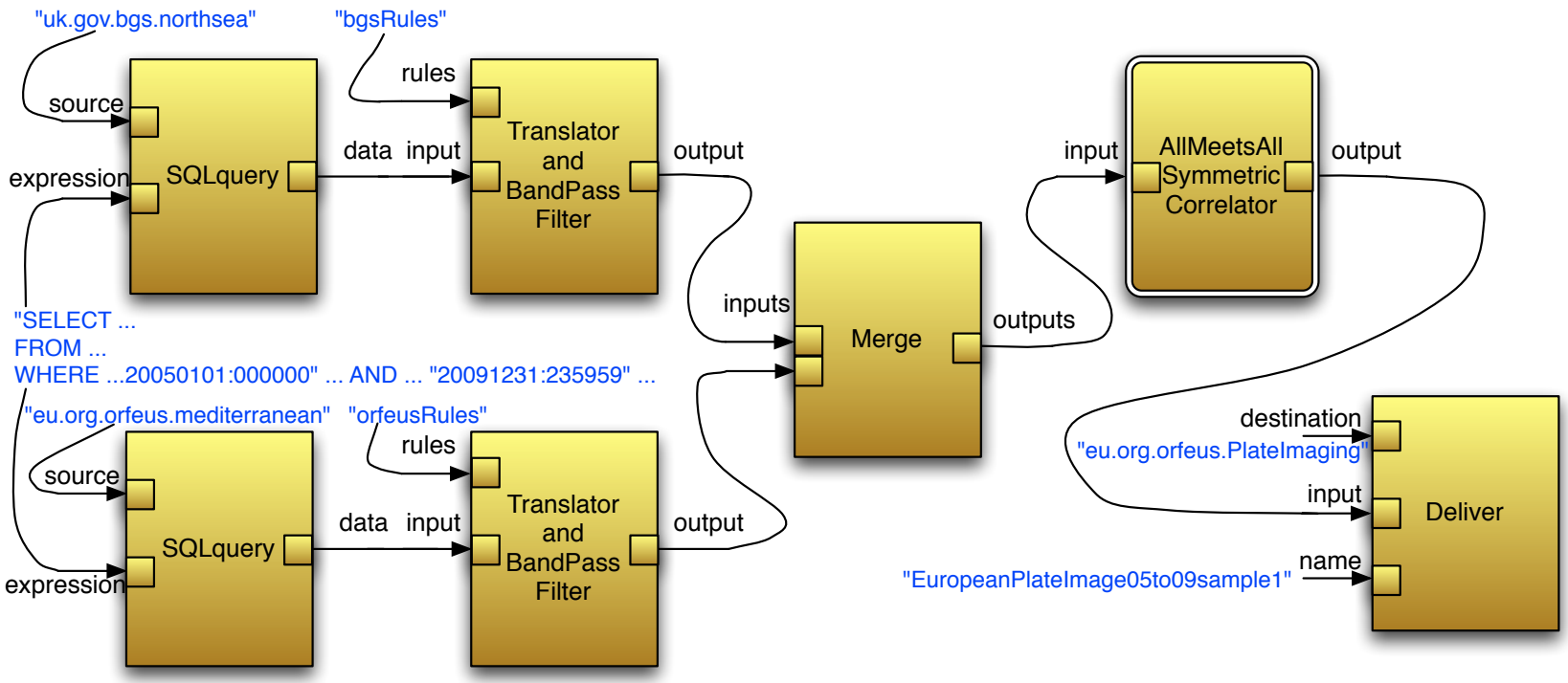
Connections (External and Internal)



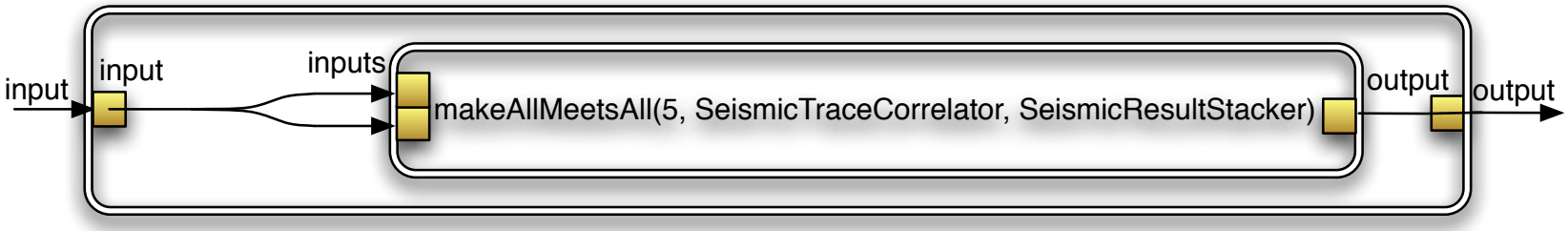
Internal Connection Interfaces

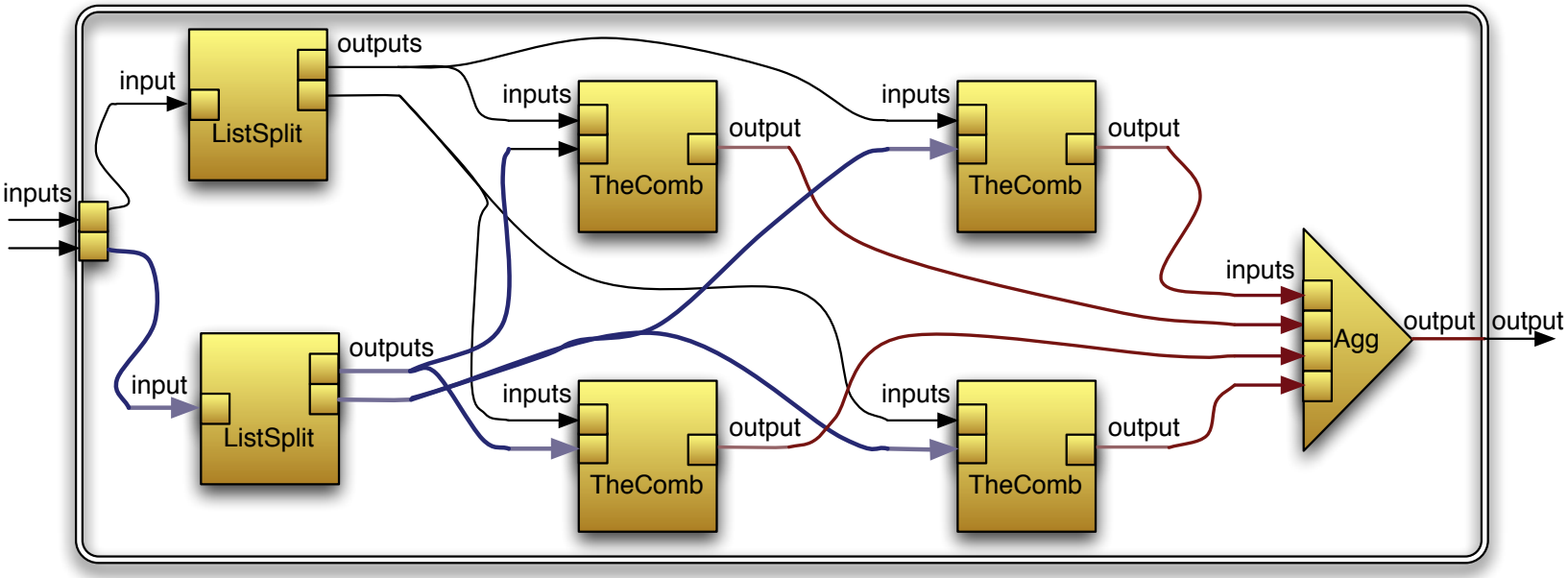
```
1
2  Type PrecociousChild is
3    PE( <> => <Connection:String::"db:Query" output> );
4
5  Type Recorder is
6    PE( <Connection:Any preserved("localhost/recorded") input> );
7
8  Type SQLQuery is
9    PE( <Connection:String::"db:Query" terminator expression;
10        Connection:String::"db:URI"   locator    resource> =>
11       <Connection:[<rest>]::"db:ResultSet" data> );
12
13 Type TupleBuild is
14   PE( <Connection:[String]::"dispel:Keys" keys;
15        Connection[]:Any::"dispel:Values"  permutable inputs> =>
16       <Connection:<rest> tuple> );
17
18 Type ListSplit is
19   PE( Stype ST is Any;
20       Dtype DT is Thing;
21       <Connection:[ST]::[DT]    input> =>
22       <Connection[]:[ST]::[DT] lockstep outputs> );
```

Composite Processing Elememnts



Composite Processing Elements





Registering New Process Elements

```
1  package eu.admire {
2    // Import existing PE from the registry.
3    use uk.org.ogsadai.SQLQuery;
4
5    // Define new PE type.
6    Type SQLToTupleList is PE( <Connection expression> =>
7                               <Connection data> );
8
9    // Define new PE function.
10   PE<SQLToTupleList> lockSQLDataSource(String dataSource) {
11     SQLQuery sqlq = new SQLQuery;
12     |-repeat enough of dataSource-| => sqlq.resource;
13     return PE( <Connection expression = sqlq.expression> =>
14                <Connection data       = sqlq.data> );
15   }
16
17   // Create new PEs.
18   PE<SQLToTupleList> SQLOnA = lockSQLDataSource("uk.org.UoE.dbA");
19   PE<SQLToTupleList> SQLOnB = lockSQLDataSource("uk.org.UoE.dbB");
20
21   // Register new entities (dependent entities will be registered as well).
22   register SQLOnA, SQLOnB;
23 }
```

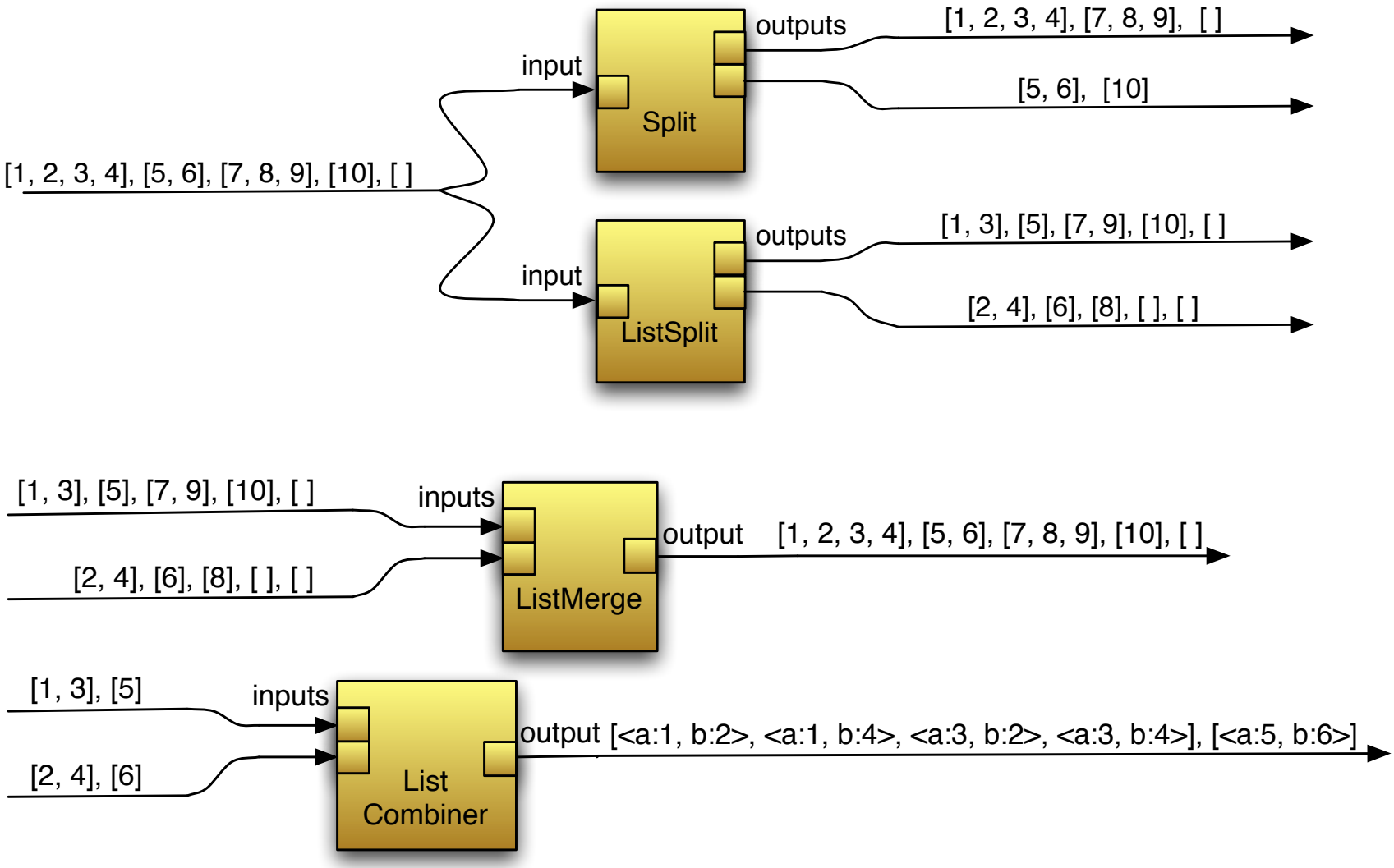
DISPEL Language Types

```
package eu.admire {
  use uk.org.ogsadai.SQLQuery;
  use uk.org.ogsadai.ListConcatenate;
  use eu.admire.madeup.DuplicatePruner;

  // CorroboratedSQLToTupleList queries multiple sources.
  Type CorroboratedSQLToTupleList is PE( <Connection expression; Connection[] permutable sources> => <Connection data> );

  // makeCorroboratedSQLQuery uses a number of parallel SQLQuery PEs to collect results for a single query.
  PE<CorroboratedSQLToTupleList> makeCorroboratedSQLQuery(Integer numberOfSources, Boolean removeDuplicates) {
    // Define internal PEs.
    SQLQuery[]      sqlq   = new SQLQuery[numberOfSources];
    ListConcatenate concat = new ListConcatenate with inputs.length = numberOfSources;

    // Input connection interfaces are defined here in advance.
    Connection   expr;
    Connection[] srcs = new Connection[numberOfSources];

    // For each data source, we need a connected SQLQuery.
    for (Integer i = 0; i < numberOfSources; i++) {
              expr => sqlq[i].expression;
          srcs[i] => sqlq[i].resource;
      sqlq[i].data => concat.input[i];
    }

    if (removeDuplicates) {
      // If removeDuplicates is true, prune duplicate responses.
      DuplicatePruner pruner = new DuplicatePruner;
      concat.output => pruner.input;
      return PE( <Connection expression = expr; Connection sources = srcs> => <Connection data = pruner.output> );
    } else {
      // Otherwise, leave as is.
      return PE( <Connection expression = expr; Connection sources = srcs> => <Connection data = concat.output> );
    }
  }

  // Register function.
  register makeCorroboratedSQLQuery;
}
```

DISPEL Structural Types

```
1  Type myPEType is PE( <Connection:String input> =>
2                       <Connection:Boolean output> );
3
4  |-1, 2, 3, 4-|:Integer => counter.input;
5
6  Stype myStype is <Integer id; String answer>;
7
8  Type myPEType is PE( <Connection:String input> =>
9                       <Connection:myStype output> );
10
11 Type myPEType is PE( <Connection:[Integer] input> =>
12                      <Connection:[Boolean] output> );
13
14 Type myPEType is PE( <Connection:Integer[][] input> =>
15                      <Connection:[Boolean[]] output> );
16
17 Type myPEType is PE( <Connection:<Boolean[][] location;
18                                  String      name> input> =>
19                      <Connection:<Integer x, y;
20                                  String  rename>   output> );
21
22 Type GridLocToAngleLoc is
23   PE( <Connection:<Integer x, y; String name> gridLoc> =>
24       <Connection:<Real angle, magnitude; String label> angleLoc> );
25
26 Stype GridLoc  is <Integer x, y; String name>;
27 Stype AngleLoc is <Real angle, magnitude; String label>;
28
29 Type GridLocToAngleLoc is PE( <Connection:GridLoc  gridLoc> =>
30                                <Connection:AngleLoc angleLoc> );
31
32 Stype GridLoc3D  is <Integer x, y, z; String name>;
33 Stype AbsGridLoc is <Integer x, y, z; Any    name>;
34 Stype GridLocXZ  is <Integer x, z;    rest>;
```

Structural Manipulation



DISPEL Domain Types

```
1  Type SQLToTupleList is
2    PE( <Connection:String::"SQLExpression"  expression> =>
3        <Connection:[<rest>]::"SQLTupleList" data> );
4
5  namespace admire "http://www.admire-project.eu/ontology#";
6
7  Type SatelliteToGreyscaleImage is
8    PE( <Connection:Image::"admire:SatelliteImage"> =>
9        <Connection:Image::"admire:GreyscaleImage"> );
10
11 Dtype "SatelliteImage" is
12   "http://www.admire-project.eu/ontology#SatelliteImage";
13 Dtype "GreyscaleImage" is
14   "http://www.admire-project.eu/ontology#GreyscaleImage";
15
16 Type SatelliteToGreyscaleImage is
17   PE( <Connection:Image::"SatelliteImage"> =>
18       <Connection:Image::"GreyscaleImage"> );
```

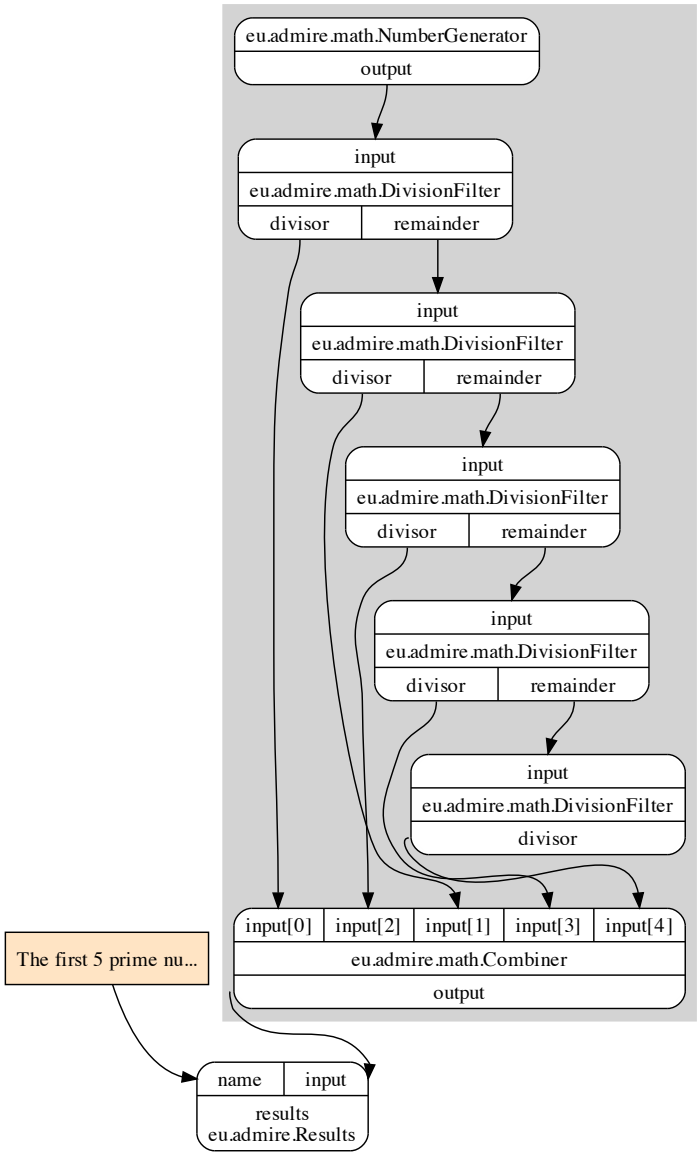
Sieve of Eratosthenes (Registration)

```
1  package eu.admire.math {
2    use eu.admire.Combiner; // Merges inputs into single stream.
3    use eu.admire.math.DivisionFilter; // Primitive filter PE.
4
5    /* Abstract PE for generating prime numbers. */
6    Type AbstractPrimeGenerator is PE( <> => <Connection:Integer::"PrimeNumber" primes> );
7
8    /* Function for constructing Sieves of Eratosthenes. */
9    PE<AbstractPrimeGenerator> makeSieveOfEratosthenes(Integer count) {
10     DivisionFilter[] filter   = new DivisionFilter[count];
11     Combiner         combiner = new Combiner with roundrobin inputs, inputs.length = count;
12
13     /* Initialise sieve stages. */
14     for (Integer i = 0; i < count - 1; i++) {
15       filter[i] = new DivisionFilter with terminator remainder;
16     }
17     filter[count - 1] = new DivisionFilter with initiator input;
18
19     /* Construct internal workflow. */
20     for (Integer i = 0; i < count - 1; i++) {
21       filter[i].remainder => filter[i + 1].input;
22       filter[i].divisible => discard;
23       filter[i].divisor   => combiner.input[i];
24     }
25     filter[count - 1].remainder => discard;
26     filter[count - 1].divisible => discard;
27     filter[count - 1].divisor   => combiner.input[count - 1];
28
29     /* Generate input until NmD (no more data) token received. */
30     |-2 to infinity-|:Integer => filter[0].input;
31
32     /* Return all prime numbers generated. */
33     return PE( <> => <Connection primes = combiner.output> );
34   }
35
36   /* Register abstract PE and PE function. */
37   register AbstractPrimeGenerator, makeSieveOfEratosthenes;
38 }
```

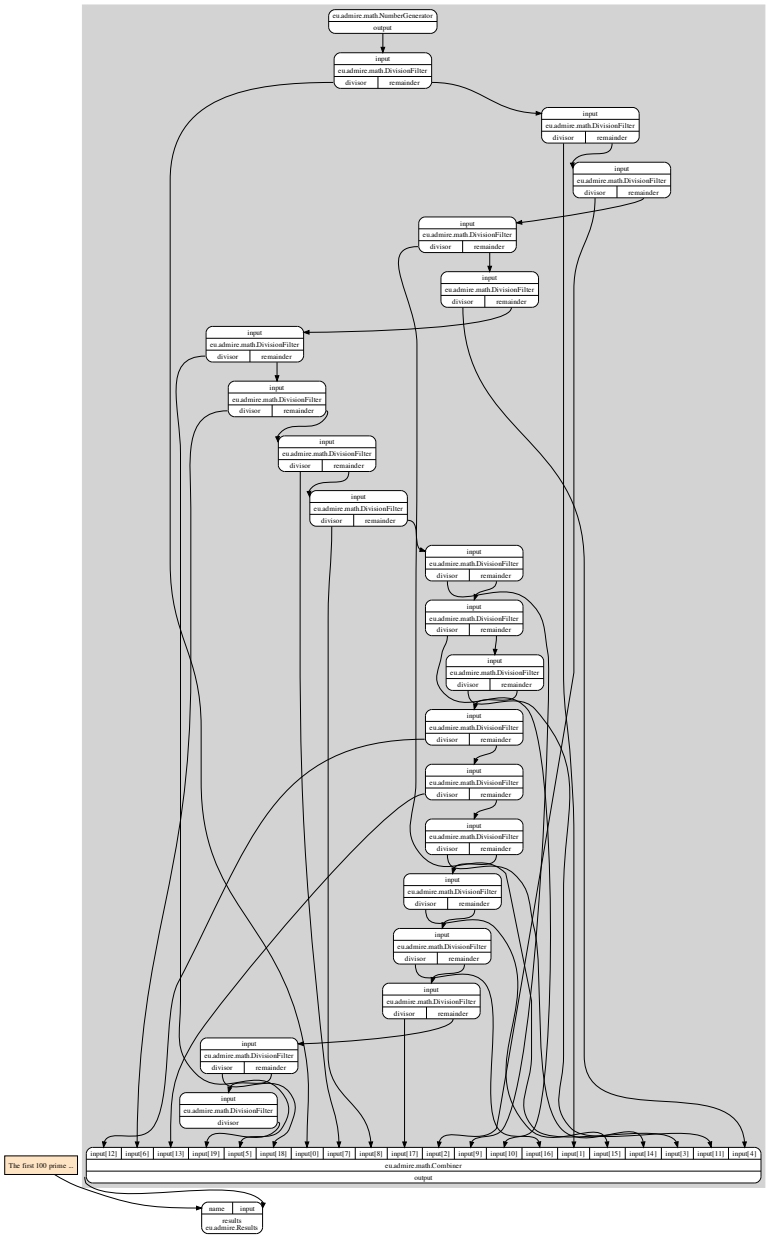
Sieve of Eratosthenes (Submission)

```
1  package eu.admire {
2    /* Import workflow components. */
3    use eu.admire.Results;
4    use eu.admire.math.AbstractPrimeGenerator;
5    use eu.admire.math.makeSieveOfEratosthenes;
6
7    /* Construct instances of PEs for workflow. */
8    PE<AbstractPrimeGenerator> SoE100 = makeSieveOfEratosthenes(100);
9    SoE100  sieve100 = new SoE100;
10   Results results  = new Results;
11
12   /* Construct the top-level workflow. */
13   |- "The first 100 prime numbers" -| => results.name;
14                          sieve100.primes => results.input;
15
16   /* Submit workflow (by submitting final component). */
17   submit results;
18 }
```

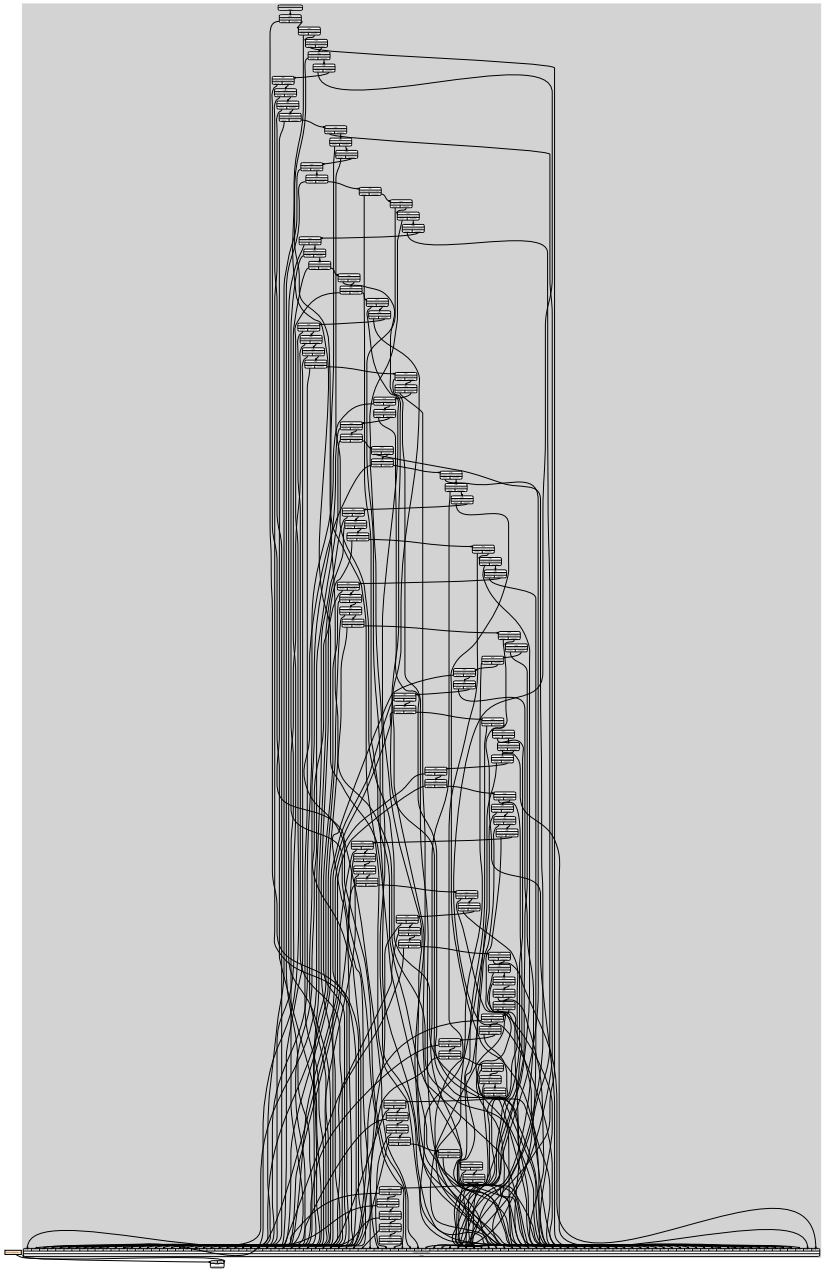
5-Element Sieve



20-Element Sieve



100-Element Sieve



k-fold Cross Validation

```
1  package eu.admire.datamine {
2    /* Import abstract types. */
3    use eu.admire.datamine.AbstractValidator;
4    use eu.admire.datamine.TrainClassifier;
5    use eu.admire.datamine.Classify;
6    /* Import sub-functions. */
7    use eu.admire.datamine.crossValidationTrain;
8    use eu.admire.datamine.crossValidationTest;
9
10   /* Produces a k-fold cross validation workflow pattern. */
11   PE<AbstractValidator>
12   makeCrossValidator(Integer count, String expression,
13                      String  class, PE<TrainClassifier> Trainer,
14                      PE<Classify>   Classifier) {
15     Connection   train;
16     Connection   test;
17     Connection[] classifiers
18       = crossValidationTrain(count, expression, train, Trainer);
19     Connection   matrix
20       = crossValidationTest(count, expression, test, classifiers,
21                             class, Classifier);
22
23     /* Return cross validation pattern. */
24     return PE( <Connection trainingData    = train;
25                 Connection testData        = test> =>
26                <Connection confusionMatrix = matrix> );
27   }
28
29   /* Register PE function. */
30   register makeCrossValidator;
31 }
```

k-fold Cross Validation

```
1  package eu.admire.datamine {
2    /* Import abstract type and sub-function. */
3    use eu.admire.datamine.TrainClassifier;
4    use eu.admire.datamine.crossValidationTrainSplit;
5
6    /* Trains classifiers for data streamed through the given connection.
7    */
8    Connection[] crossValidationTrain(Integer    count,
9                                      String     expression,
10                                     Connection input,
11                                     PE<TrainClassifier> Trainer) {
12     /* Split the training data for each fold. */
13     Connection[] split
14       = crossValidationTrainSplit(count, expression, input);
15     Connection[] results = new Connection[count];
16
17     /* Train a classifier for each training set. */
18     for (Integer i = 0; i < count; i++) {
19       Trainer train = new Trainer;
20       split[i] => train.data;
21       train.classifier => results[i];
22     }
23
24     /* Return the connections through which each classifier will be
25     streamed. */
26     return results;
27   }
28
29   /* Register function. */
30   register crossValidationTrain;
31 }
```

k-fold Cross Validation

```
1  package eu.admire.datamine {
2    /* Import OGSA-DAI activities. */
3    use uk.org.ogsadai.TupleArithmeticSample;
4    use uk.org.ogsadai.TupleUnionAll;
5
6    /* Produces training sets from data streamed through the given
     connection. */
7    Connection[] crossValidationTrainSplit(Integer    count,
8                                           String     expression,
9                                           Connection input) {
10     /* Data needs to be divided and resampled for each fold. */
11     TupleArithmeticSample sample  = new TupleArithmeticSample
12       with result.length = count;
13     TupleUnionAll[]       union   = new TupleUnionAll[count];
14     Connection[]          outputs = new Connection[count];
15
16     /* Sample the input data into multiple subsets. */
17     |- expression -| => sample.expression;
18                 input => sample.data;
19
20     /* Create training sets; leave out one sample each time. */
21     for (Integer i = 0; i < count; i++) {
22       union[i] = new TupleUnionAll with data.length = count - 1;
23       for (Integer j = 0; j < i; j++) {
24         sample.result[j] => union[i].data[j];
25       }
26       for (Integer j = i + 1; j < count; j++) {
27         sample.result[j] => union[i].data[j - 1];
28       }
29       union[i].data => outputs[i];
30     }
31
32     /* Return the connections through which training sets will be
       streamed. */
33     return outputs;
34   }
35
36   /* Register function. */
37   register crossValidationTrainSplit;
38 }
```

k-fold Cross Validation

```
1  package eu.admire.datamine {
2    /* Import abstract type and sub-function. */
3    use eu.admire.datamine.Classify;
4    use eu.admire.datamine.crossValidationTestSplit;
5    /* Import OGSA-DAI activities. */
6    use uk.org.ogsadai.TupleUnionAll;
7    use uk.org.ogsadai.TupleSimpleMerge;
8    use uk.org.ogsadai.GroupBy;
9
10   /* Tests classifiers for data streamed through the given connection.
     */
11   Connection crossValidationTest(Integer      count,
12                                  String       expression,
13                                  Connection   input,
14                                  Connection[] classifiers,
15                                  String       class,
16                                  PE<Classify> Classifier) {
17     Connection[]  split
18       = crossValidationTestSplit(count, expression, input);
19     Connection    result;
20     TupleUnionAll union = new TupleUnionAll
21       with data.length  = count;
22
23     /* Apply and compare each classifier to a test data set. */
24     for (Integer i = 0; i < count; i++) {
25       Classifier       classify = new Classifier;
26       TupleSimpleMerge merge    = new TupleSimpleMerge;
27       GroupBy          groupBy  = new GroupBy;
28
29       /* Feed inputs into classifier instances. */
30       classifiers[i] => classify.classifier;
31       |- "predicted" -| => classify.resultColumnName;
```

k-fold Cross Validation

```
1         split[i] => classify.data;
2
3         /* Merge the test data with predicted classifications. */
4                 split[i] => merge.data[0];
5         classify.result => merge.data[1];
6
7         /* Count the expected and actual results. */
8                           merge.result => groupBy.data;
9         |- [ class, "predicted" ] -| => groupBy.columnIds;
10                  |- [ "COUNT(*)" ] -| => groupBy.aggregates;
11        |- [ "expected", "actual", "count" ] -|
12                                       => groupBy.resultColumnNames;
13                        groupBy.result => union.data[i];
14    }
15
16    /* Sum up the results for each classifier and list the results. */
17    GroupBy groupBy = new GroupBy;
18                            union.data => groupBy.data;
19    |- [ "expected", "actual" ] -| => groupBy.columnIds;
20            |- [ "SUM(count)" ] -| => groupBy.aggregates;
21    |- [ "expected", "actual", "count" ] -|
22                                       => groupBy.resultColumnNames;
23                        groupBy.result => result;
24
25    /* Return the connection through which results will be streamed.
      */
26    return result;
27  }
28
29  /* Register function. */
30  register crossValidationTest;
31 }
```

k-fold Cross Validation

```
1  package eu.admire.datamine {
2    /* Import OGSA-DAI activities. */
3    use uk.org.ogsadai.TupleArithmeticSample;
4
5    /* Produces test data sets from data streamed through the con-
     nection. */
6    Connection[] crossValidationTestSplit(Integer    count,
7                                          String     expression,
8                                          Connection input) {
9      TupleArithmeticSample sample = new TupleArithmeticSample
10       with result.length = count;
11     Connection[] outputs = new Connection[count];
12
13     /* Sample the input data into multiple subsets. */
14     |- expression -| => sample.expression;
15                input => sample.data;
16
17     /* Create test sets. */
18     for (Integer i = 0; i < count; i++) {
19       sample.result[i] => outputs[i];
20     }
21
22     /* Return the connections through which each test set will be
       streamed. */
23     return outputs;
24   }
25
26   /* Register function. */
27   register crossValidationTestSplit;
28 }
```

k-fold Cross Validation Submission

```
1  package eu.admire.datamine {
2    /* Import abstract types. */
3    use eu.admire.datamine.TrainClassifier;
4    use eu.admire.datamine.Classify;
5    /* Import implementable PEs. */
6    use uk.org.ogsadai.SQLQuery;
7    use eu.admire.Results;
8    use eu.admire.BuildClassifier;
9    use eu.admire.datamine.AbstractLearningTest;
10   use eu.admire.datamine.makeCrossValidator;
11
12   /* Constructs a classifier trainer from an existing PE. */
13   PE<TrainClassifier> createBuildClassifier() {
14     BuildClassifier train = new BuildClassifier;
15
16     /* Input default values for unused interfaces. */
17     |- "play" -| => train.classIndex;
18     |- [ "outlook", "windy", "play" ] -| => train.columnIndices;
19     |- [ ["sunny", "rainy", "overcast"], ["true", "false"], [ "yes", "no" ] ] -| => train.nominalValues;
20
21     /* Return wrapper for BuildClassifier. */
22     return TrainClassifier(<Connection data=train.data> => <Connection classifier=train.classifier>);
23   }
24
25   /* Set inputs. */
26   String  expression = "select outlook, temperature, humidity, windy, play from tennis order by day";
27   String  resource   = "MySQLResource";
28   Integer count      = 4;
29   String  splitExpression =  "Floor(Rand(54783) * " + count + ")";
30
31   /* Extract training data. */
32   SQLQuery queryTrain = new SQLQuery;
33   |- expression -| => queryTrain.expression;
34   |- resource -| => queryTrain.resource;
35
36   /* Extract test data. */
37   SQLQuery queryTest = new SQLQuery;
38   |- expression -| => queryTest.expression;
39   |- resource -| => queryTest.resource;
40
41   /* Construct a simple classifier trainer. */
42   PE<TrainClassifier> Trainer = createBuildClassifier();
43
44   /* Construct a cross validator. */
45   PE<AbstractLearningTest> CrossValidator = makeCrossValidator(count, splitExpression, "play", Trainer, Classify);
46   CrossValidator validator = new CrossValidator;
47   Results results          = new Results;
48
49   /* Construct top-level workflow. */
50   queryTrain.data => validator.trainingData;
51   queryTest.data => validator.testingData;
52   |- "result" -| => results.name;
53   validator.confusionMatrix => results.input;
54
55   /* Submit workflow. */
56   submit validator;
57 }
```

Example of k-fold Cross Validation