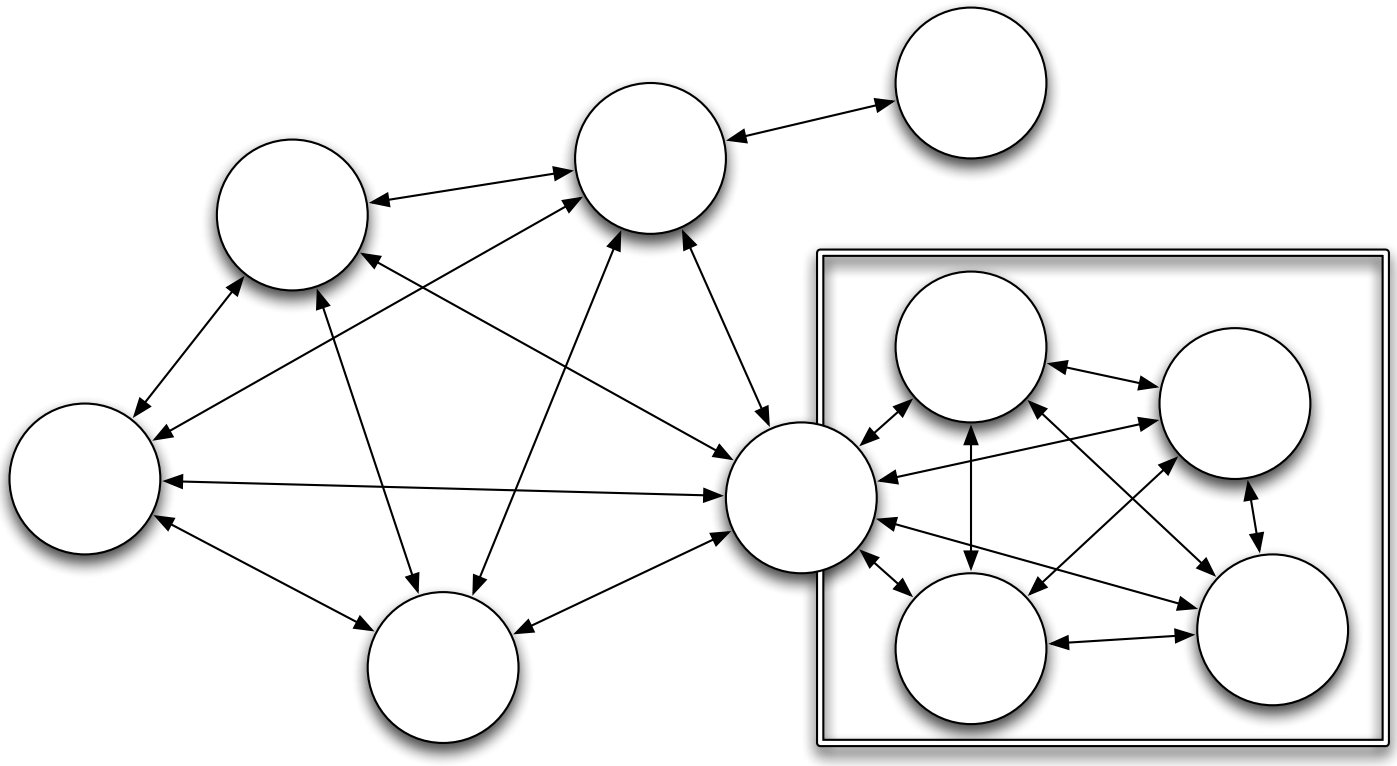
Dispel Tutorial

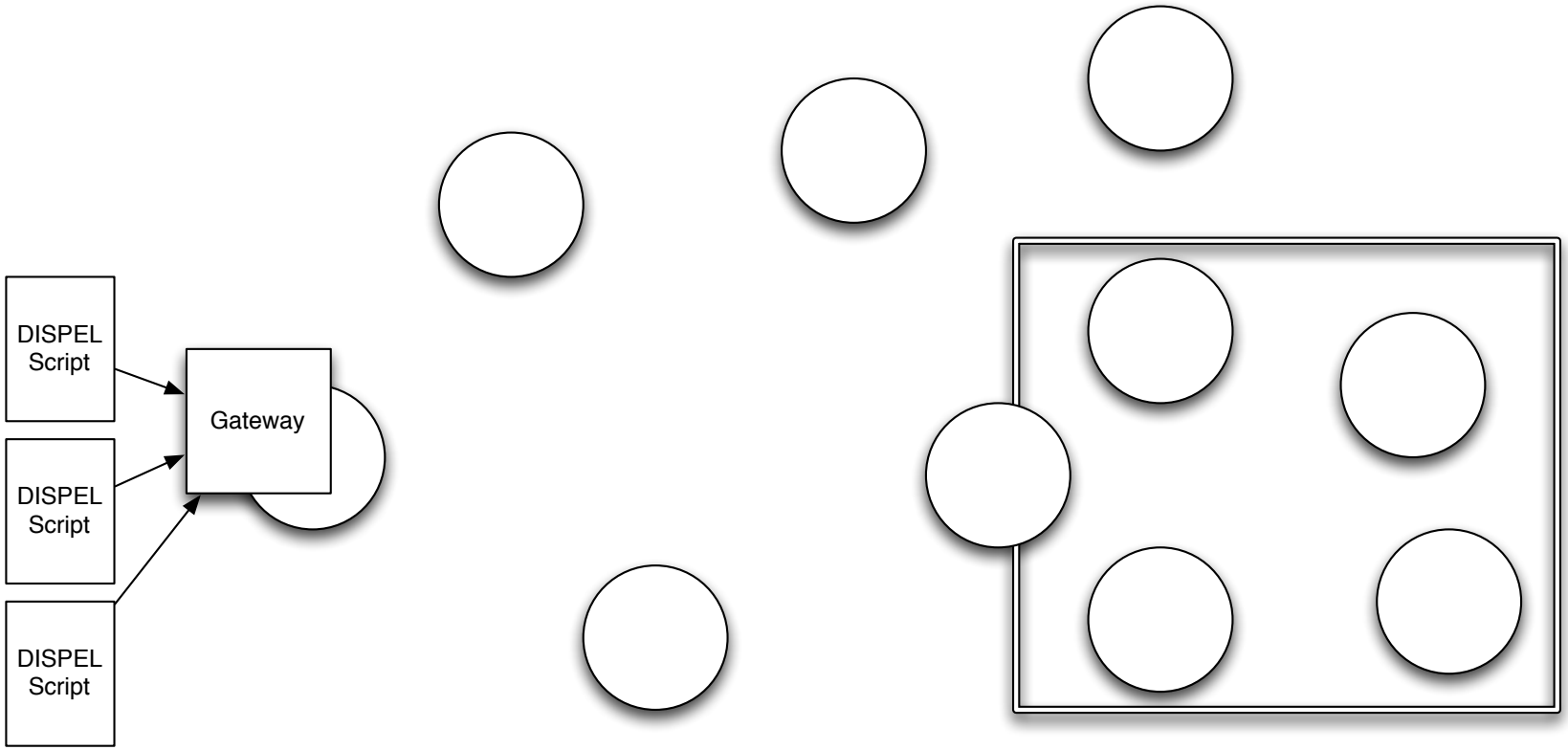
Paul W Martin

July 16, 2012

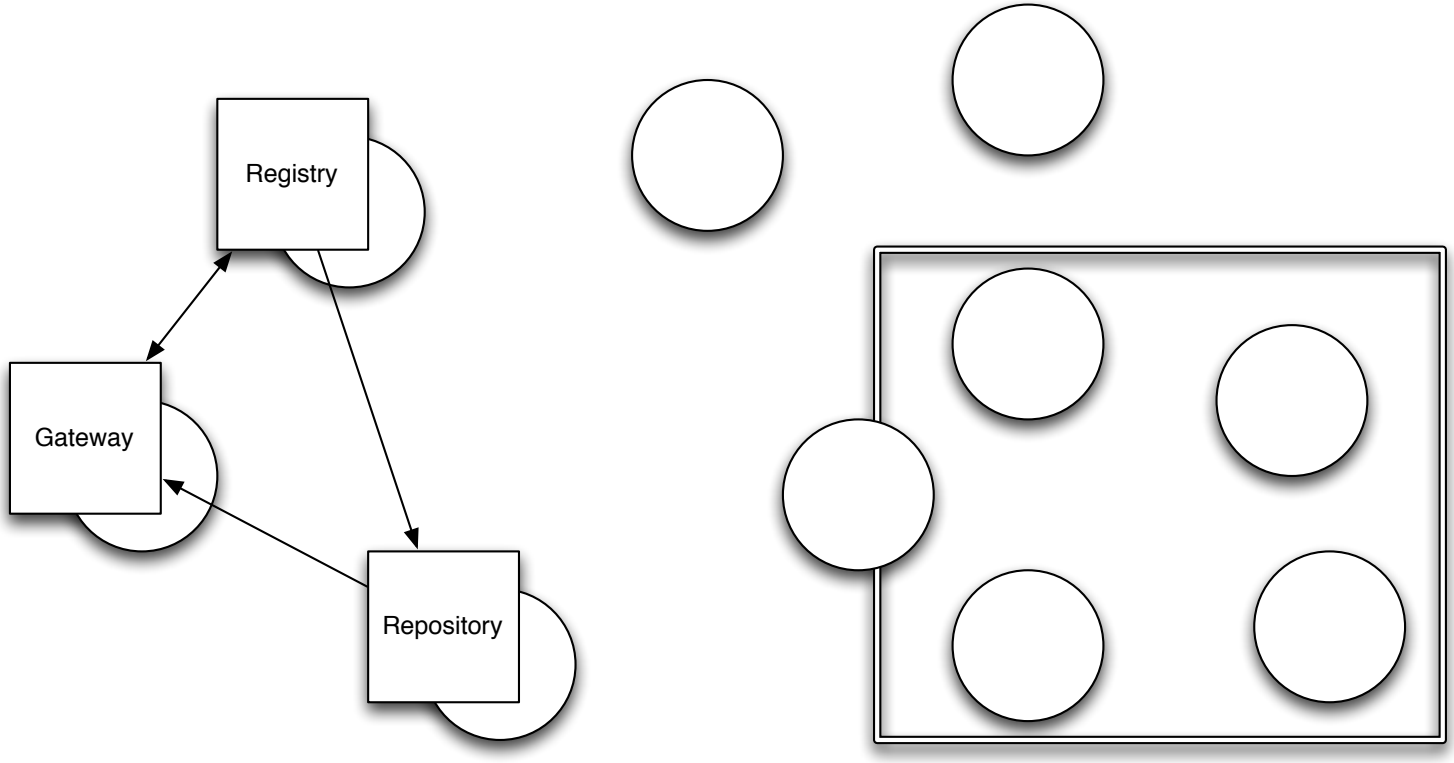
Workflow enactment



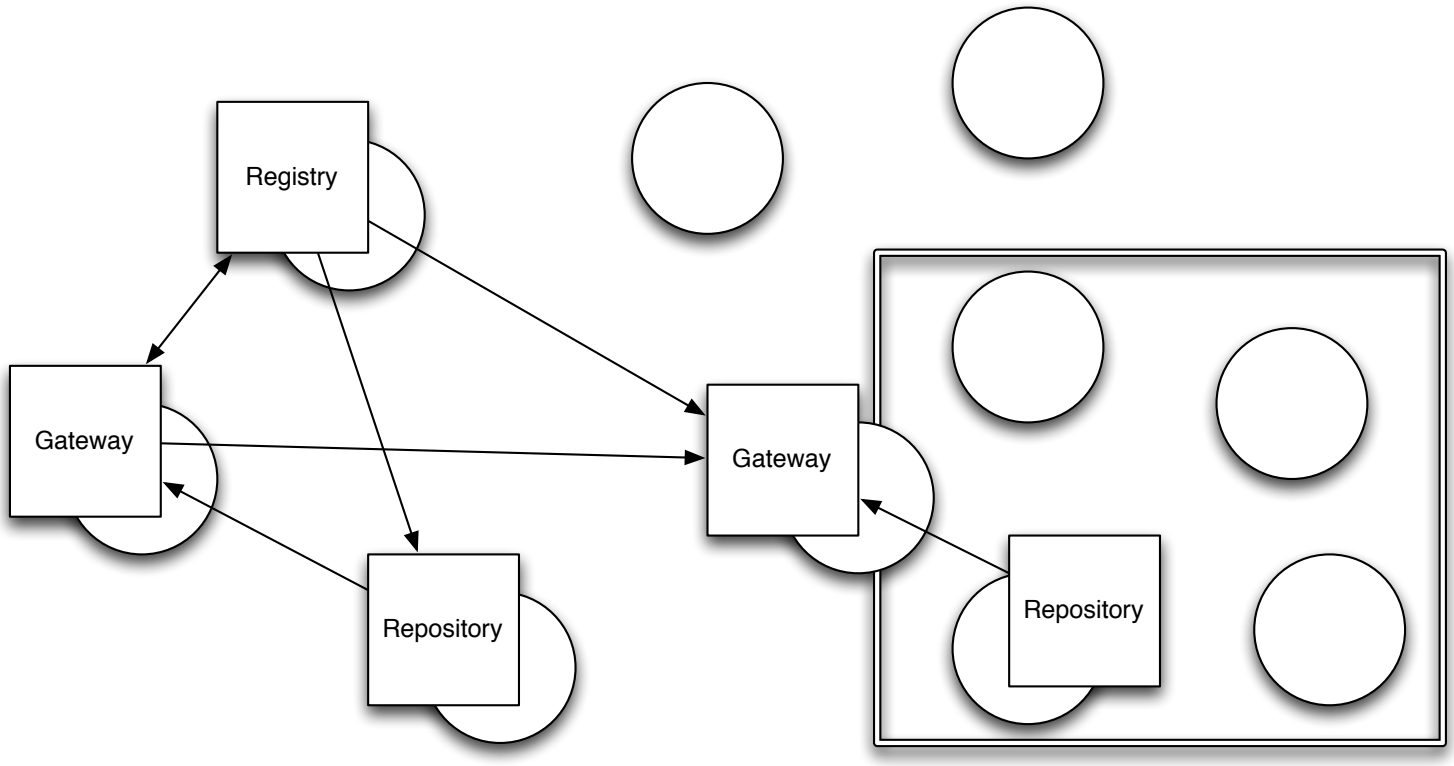
Workflow enactment



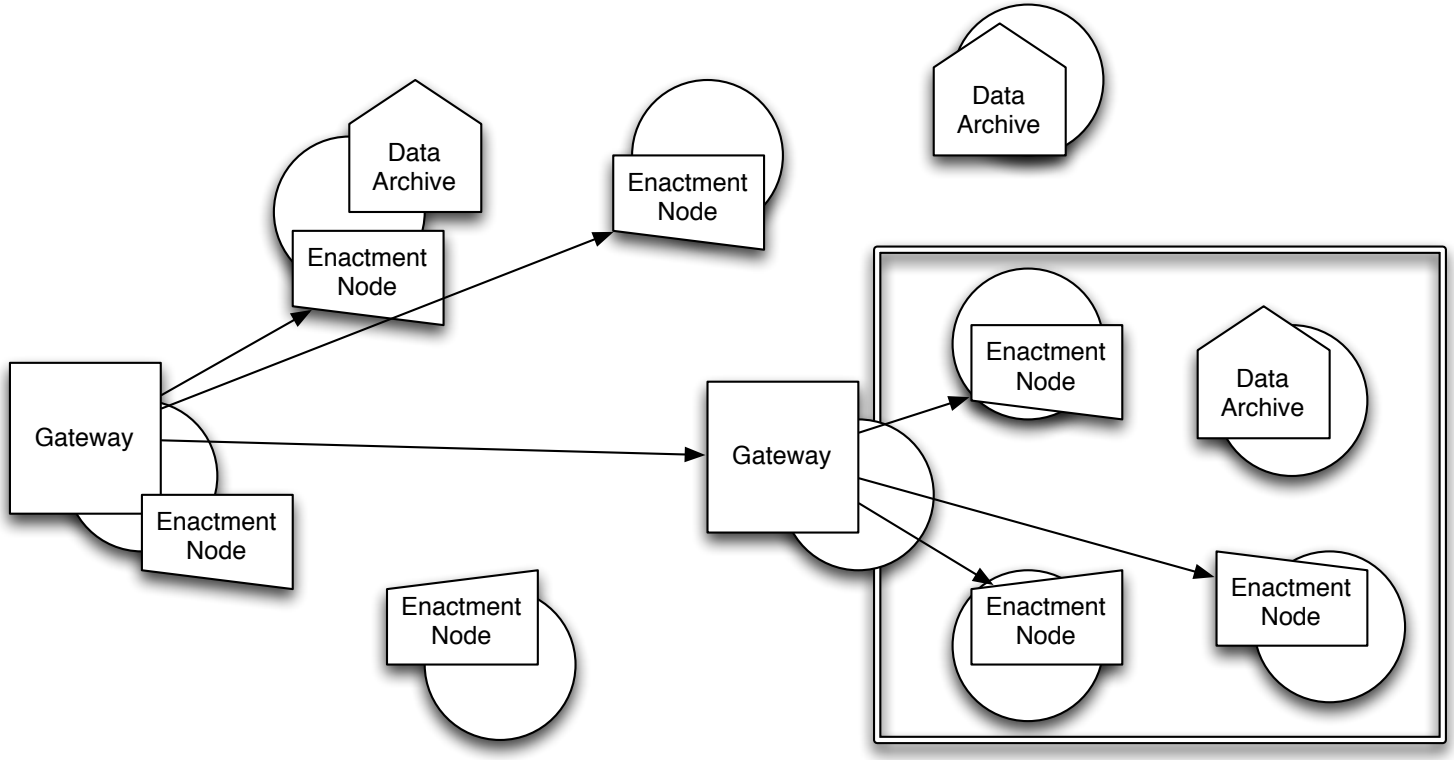
Workflow enactment



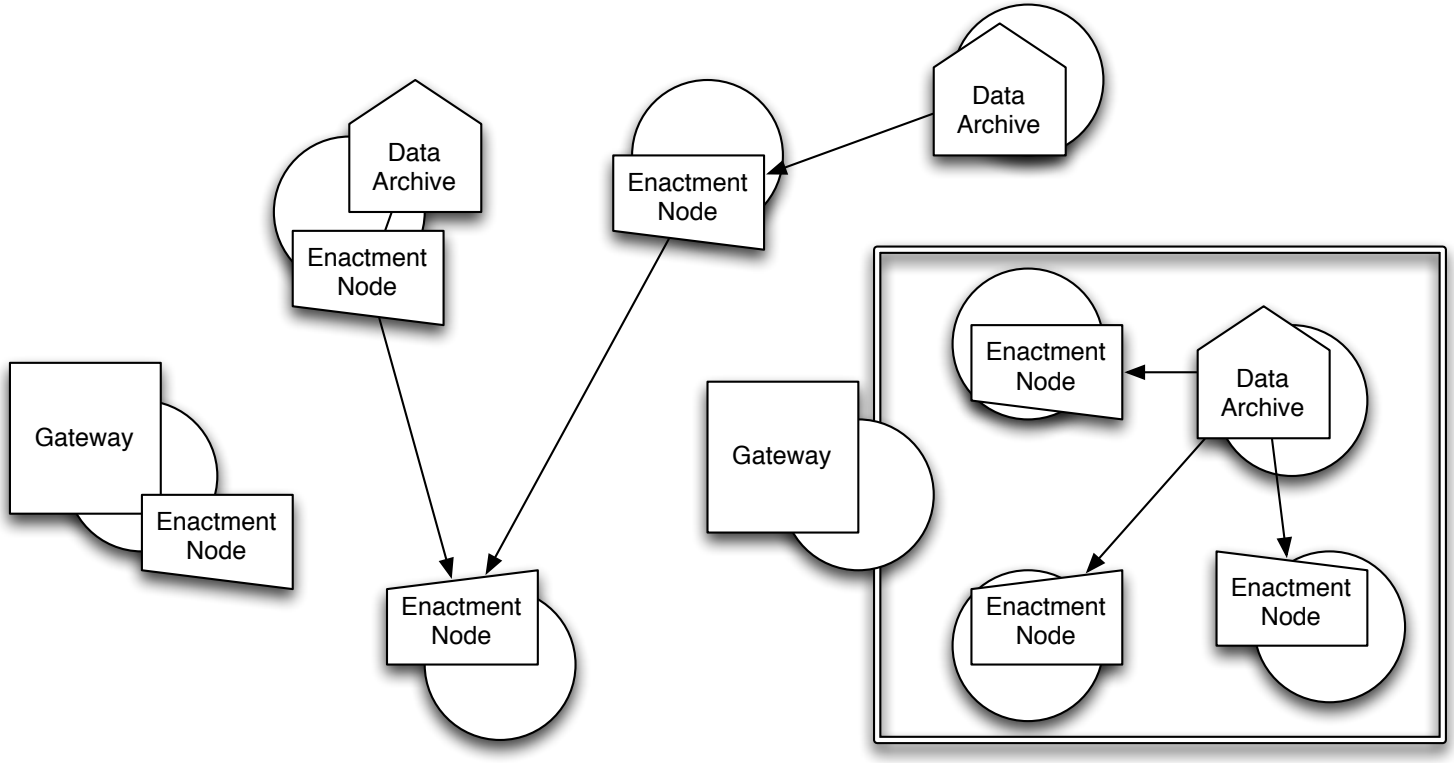
Workflow enactment



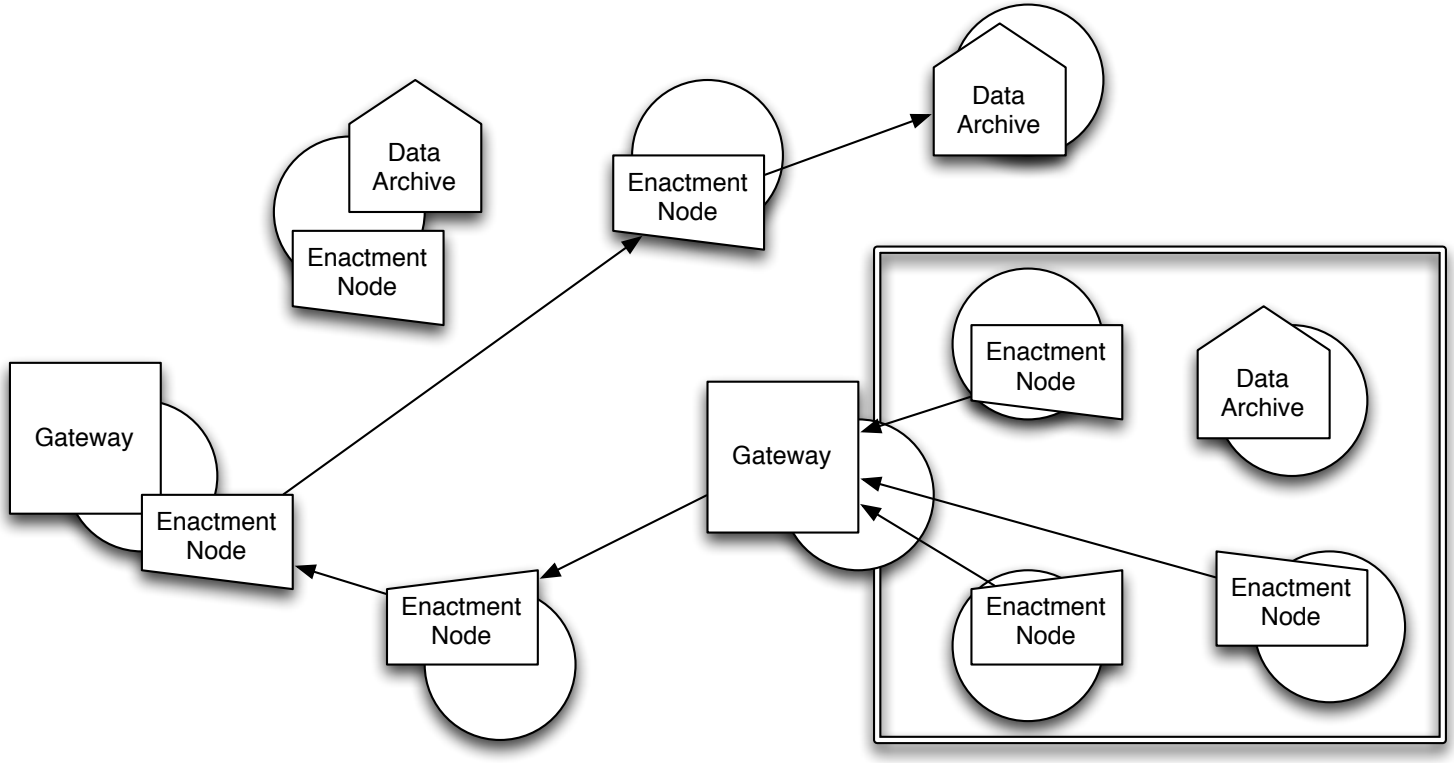
Workflow enactment



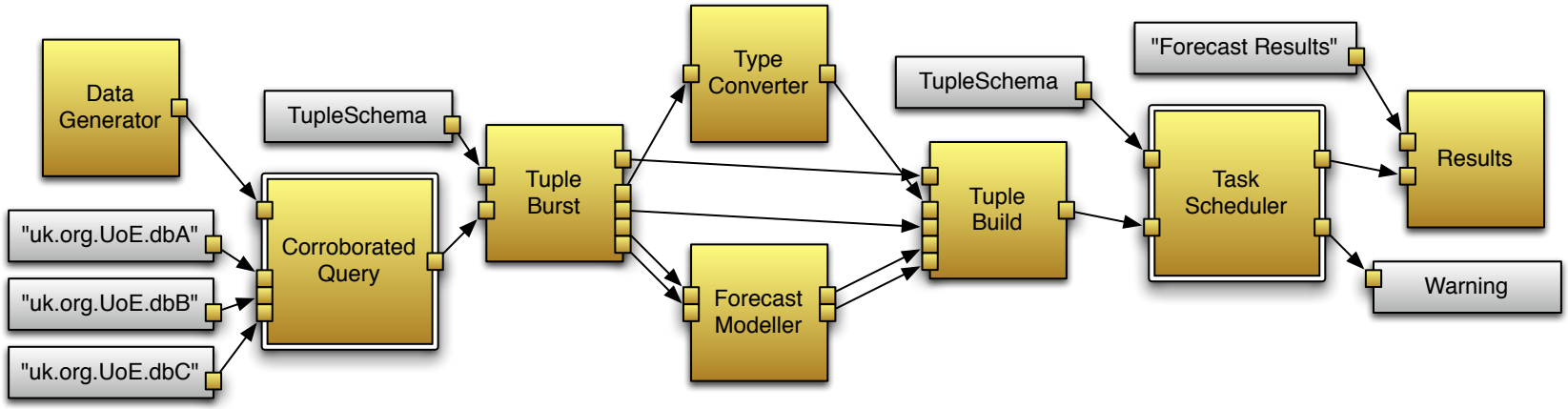
Workflow enactment



Workflow enactment



Workflow


Data Generator
TupleSchema
"uk.org.UoE.dbA"
"uk.org.UoE.dbB"
"uk.org.UoE.dbC"
Corroborated Query
Tuple Burst
Type Converter
Forecast Modeller
TupleSchema
Tuple Build
"Forecast Results"
Task Scheduler
Results
Warning

A simple script

```
1  // Import PEs from the local registry.
2  use dispel.db.SQLQuery;
3  use dispel.core.Results;
4
5  // Create instances of PEs.
6  SQLQuery query   = new SQLQuery;
7  Results  results = new Results;
8
9  // Construct workflow and feed in data.
10 |-"SELECT * FROM littleblackbook WHERE id <= 10"-| => query.expression;
11 |-"uk.org.UoE.dbA"-| => query.resource;
12 |-"10 entries from the little black book"-| => results.name;
13 query.data => results.input;
14
15 // Submit the entire workflow.
16 submit results;
```

Stream literals

```
String query1 = "SELECT name FROM littleblackbook WHERE id <= 10";
String query2 = "SELECT name FROM littleblackbook"
                + "WHERE id > 10 AND id <= 20";
String query3 = "SELECT address FROM littleblackbook"
                + "WHERE name = 'David Hume'";

|-query1, query2, query3-| => query.expression;


String database1 = "uk.org.UoE.dbA";
String database2 = "uk.org.UoE.dbB";
String database3 = "uk.org.UoE.dbC";

|-database1, database2, database3-| => query.resource;
```

Stream literals

```
String query1 = "SELECT name FROM littleblackbook WHERE id <= 10";
String query2 = "SELECT name FROM littleblackbook"
                + "WHERE id > 10 AND id <= 20";
String query3 = "SELECT address FROM littleblackbook"
                + "WHERE name = 'David Hume'";

|-query1, query2, query3-| => query.expression;

|-repeat 3 of "uk.org.UoE.dbA"-| => query.resource;
```

Stream literals

```
String query1 = "SELECT name FROM littleblackbook WHERE id <= 10";
String query2 = "SELECT name FROM littleblackbook"
                + "WHERE id > 10 AND id <= 20";
String query3 = "SELECT address FROM littleblackbook"
                + "WHERE name = 'David Hume'";

|-query1, query2, query3-| => query.expression;

|-repeat enough of "uk.org.UoE.dbA"-| => query.resource;
```

A precocious script

```
// Import PEs from the local registry.
use dispel.db.SQLQuery;
use dispel.tutorial.PrecociousChild;
use dispel.core.Results;

// Create instances of PEs.
PrecociousChild child   = new PrecociousChild;
SQLQuery        query   = new SQLQuery;
Results         results = new Results;

// Construct workflow and feed in data.
child.output => query.expression;
|-repeat enough of "uk.org.UoE.dbA"-| => query.response;
|-"Adult responses"-| => results.name;
query.data => results.input;

// Submit the entire workflow.
submit results;
```

Abstract PE types

```
PE( [Declarations]
    <Input_1, ..., Input_m> => <Output_1, ..., Output_n> )
    [with Properties]
```

```
Type SQLQuery is
  PE( <Connection:String::"db:SQLQuery" terminator expression;
       Connection:String::"db:URI"      locator    resource> =>
      <Connection:[<rest>]::"db:TupleRowSet" data> )
```

Abstract PE types

```
PE( [Declarations]
    <Input_1, ..., Input_m> => <Output_1, ..., Output_n> )
    [with Properties]
```

```
Type SortedListMerge is
  PE( Stype List   is [Any];
      Dtype Domain is Thing;
      <Connection[]:List::Domain permutable inputs> =>
      <Connection:  List::Domain output> )
  with @description = "Merges each round of inputs according to a"
                    + "standard ordering.";
```

Abstract PE types

```
PE( [Declarations]
    <Input_1, ..., Input_m> => <Output_1, ..., Output_n> )
    [with Properties]
```

```
Type SQLToTupleList is
    PE( <Connection:String::"db:SQLQuery" expression> =>
        <Connection:[<rest>]::"db:TupleRowSet" data> );
```

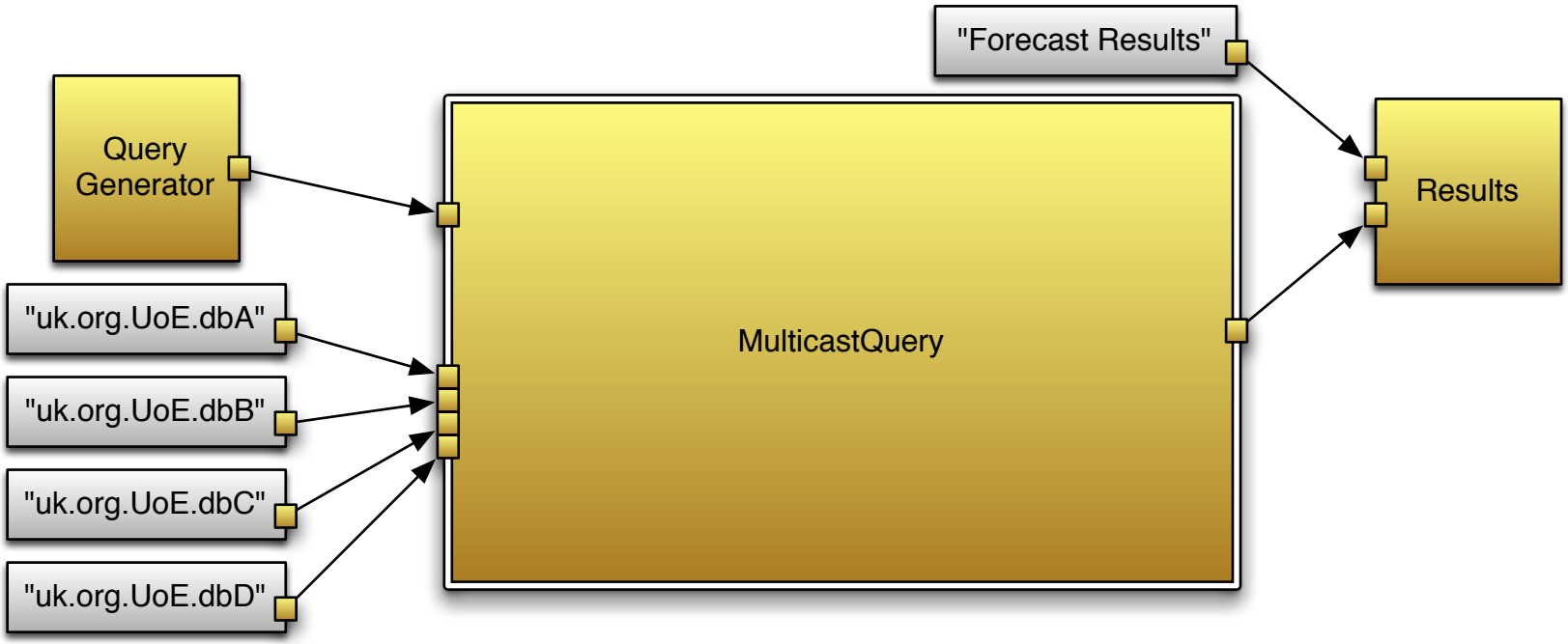
A simple composite PE

```
1   package tutorial.example {
2     // Import existing PE from the registry.
3     use dispel.db.SQLQuery;
4
5     // Define new PE type.
6     Type SQLToTupleList is PE( <Connection expression> =>
7                                <Connection data> );
8
9     // Define new PE constructor.
10    PE<SQLToTupleList> lockSQLDataSource(String dataSource) {
11      SQLQuery query = new SQLQuery;
12      |-repeat enough of dataSource-| => query.source;
13      return PE( <Connection expression = query.expression> =>
14                 <Connection data       = query.data> );
15    }
16
17    // Create new PEs.
18    PE<SQLToTupleList> TutorialQuery = lockSQLDataSource("uk.org.UoE.dbA");
19    PE<SQLToTupleList> MirrorQuery   = lockSQLDataSource("uk.org.UoE.dbB");
20
21    // Register new entities.
22    register TutorialQuery, MirrorQuery;
23  }
```

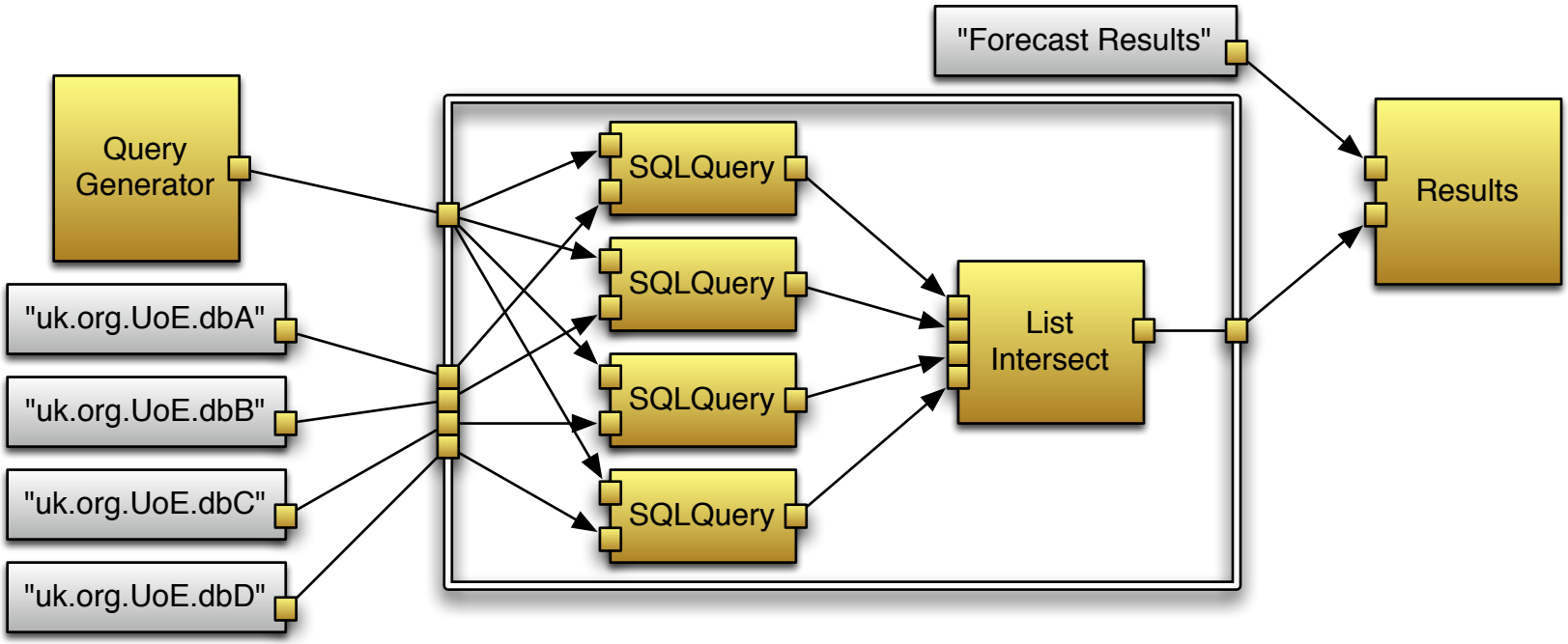
Using the composite PE

```
1  // Import PEs from the local registry.
2  use tutorial.example.TutorialQuery;
3  use dispel.core.Results;
4
5  // Create instances of PEs (no import necessary).
6  TutorialQuery query   = new TutorialQuery;
7  Results       results = new Results;
8
9  // Connect PEI together to create workflow (no data source needed).
10 |-"SELECT * FROM littleblackbook WHERE id <= 10"-| => query.expression;
11 |-"10 entries from the little black book"-| => results.name;
12 query.data => results.input;
13
14 // Submit workflow.
15 submit results;
```

Multicasting with composite PEs



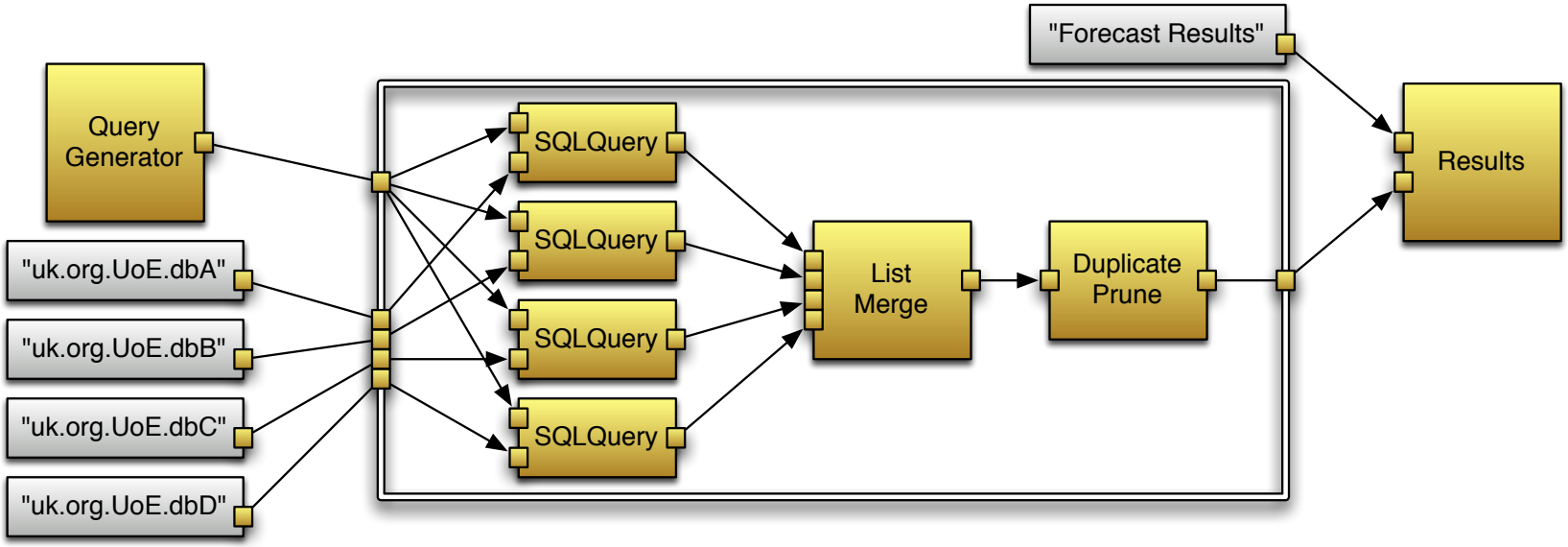
Multicasting with composite PEs



A PE constructor featuring iteration

```
1  package tutorial.example {
2    use dispel.db.SQLQuery;
3    use dispel.list.ListIntersect;
4
5    // A PE type which queries multiple sources.
6    Type MulticastQuery is PE( <Connection expression; Connection[] sources> => <Connection data> );
7
8    // Use parallel SQLQuery instances to corroborate results.
9    PE<MulticastQuery> makeCorroboratedSQLQuery(Integer numberOfSources) {
10     // Define aliases for workflow inputs in advance.
11     Connection   expr;
12     Connection[] srcs = new Connection[numberOfSources];
13     // Create instances of internal PEs.
14     SQLQuery[]     queries   = new SQLQuery[numberOfSources];
15     ListIntersect intersect = new ListIntersect with inputs.length = numberOfSources;
16
17     // Connect SQLQuery instances in parallel.
18     for (Integer i = 0; i < numberOfSources; i++) {
19       queries[i] = new SQLQuery;
20                   expr => queries[i].expression;
21                 srcs[i] => queries[i].resource;
22       queries[i].data => intersect.inputs[i];
23     }
24
25     // Return intersection of query results.
26     return PE( <Connection expression = expr; Connection[] sources = srcs> =>
27                <Connection data = intersect.output> );
28   }
29
30   register MulticastQuery, makeCorroboratedSQLQuery;
31 }
```

Multicasting with composite PEs



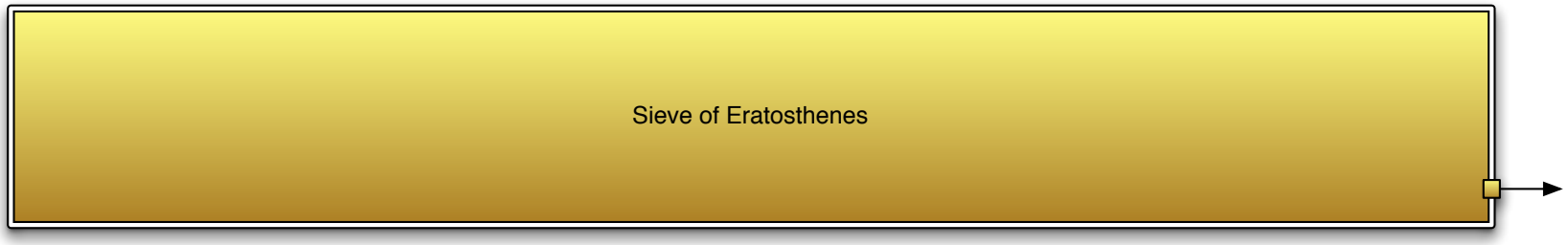
A PE constructor featuring selection

```
1  package tutorial.example {
2    use dispel.db.SQLQuery;
3    use dispel.list.ListConcatenate;
4    use dispel.list.ListUnion;
5    use dispel.core.DuplicatePrune;
6
7    // A PE type which queries multiple sources.
8    Type MulticastQuery is PE( <Connection expression; Connection[] sources> => <Connection data> );
9
10   // Use parallel SQLQuery instances to collect results.
11   PE<MulticastQuery> makeMassSQLQuery(Integer numberOfSources, Boolean removeDuplicates) {
12     // Prepare components for connection.
13     Connection   expr;
14     Connection[] srcs    = new Connection[numberOfSources];
15     SQLQuery[]   queries = new SQLQuery[numberOfSources];
16     ListMerge    merge   = new ListMerge;
17
18     // Connect SQLQuery instances in parallel.
19     for (Integer i = 0; i < numberOfSources; i++) {
20       queries[i] = new SQLQuery;
21                  expr => queries[i].expression;
22               srcs[i] => queries[i].resource;
23       queries[i].data => merge.inputs[i];
24     }
25
26     if (removeDuplicates) {
27       // If removeDuplicates is true, prune duplicate responses.
28       DuplicatePrune prune = new DuplicatePrune;
29       merge.output => prune.input;
30       return PE( <Connection expression = expr; Connection[] sources = srcs> => <Connection data = prune.output> );
31     } else {
32       // Otherwise, leave as is.
33       return PE( <Connection expression = expr; Connection[] sources = srcs> => <Connection data = merge.output> );
34     }
35   }
36
37   register makeMassSQLQuery;
38 }
```

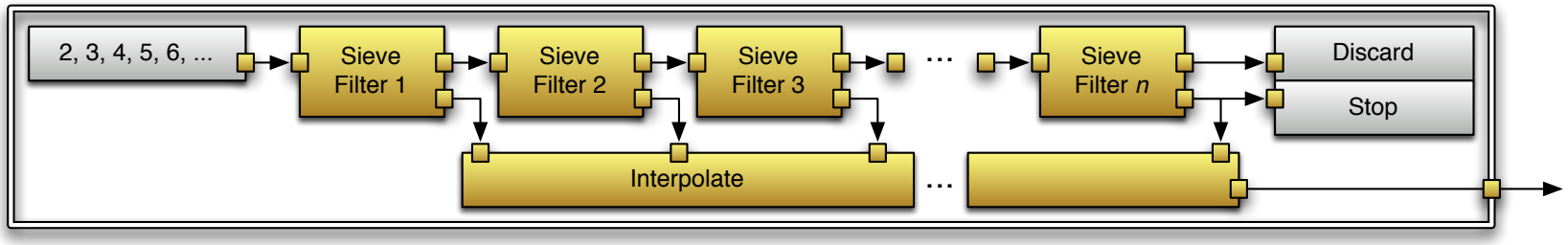
PE type specification

```
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> );
```

Sieve of Eratosthenes



Sieve of Eratosthenes



Sieve of Eratosthenes

```
1  package tutorial.example {
2    // Import filters.
3    use dispel.filter.AbstractFilter;
4    use dispel.filter.HeadFilter;
5    use dispel.filter.ProgrammableIntegerFilter;
6
7    // Define sieve element constructor.
8    PE<AbstactFilter> makeSieveElement() {
9      // Create reference to input connection.
10     Connection:Integer input;
11     // Instantiate internal components.
12     HeadFilter split = new HeadFilter;
13     ProgrammableIntegerFilter divide = new ProgrammableIntegerFilter with parameters.length = 1;
14
15     // Construct internal workflow.
16     |-"x if (x % $0) == 0"-| => divide.expression;
17     input                   => split.input;
18     split.head              => divide.parameter[0];
19     split.tail              => divide.input;
20     divide.unfiltered       => discard;
21
22     // Output first integer received and all indivisible integers.
23     return PE( <Connection input = input> =>
24                <Connection filtered = split.head; Connection unfiltered = divide.filtered> );
25   }
26
27   // Create the sieve element PE.
28   PE<AbstractFilter> SieveElement = makeSieveElement();
29
30   // Register sieve element.
31   register SieveElement;
32 }
```

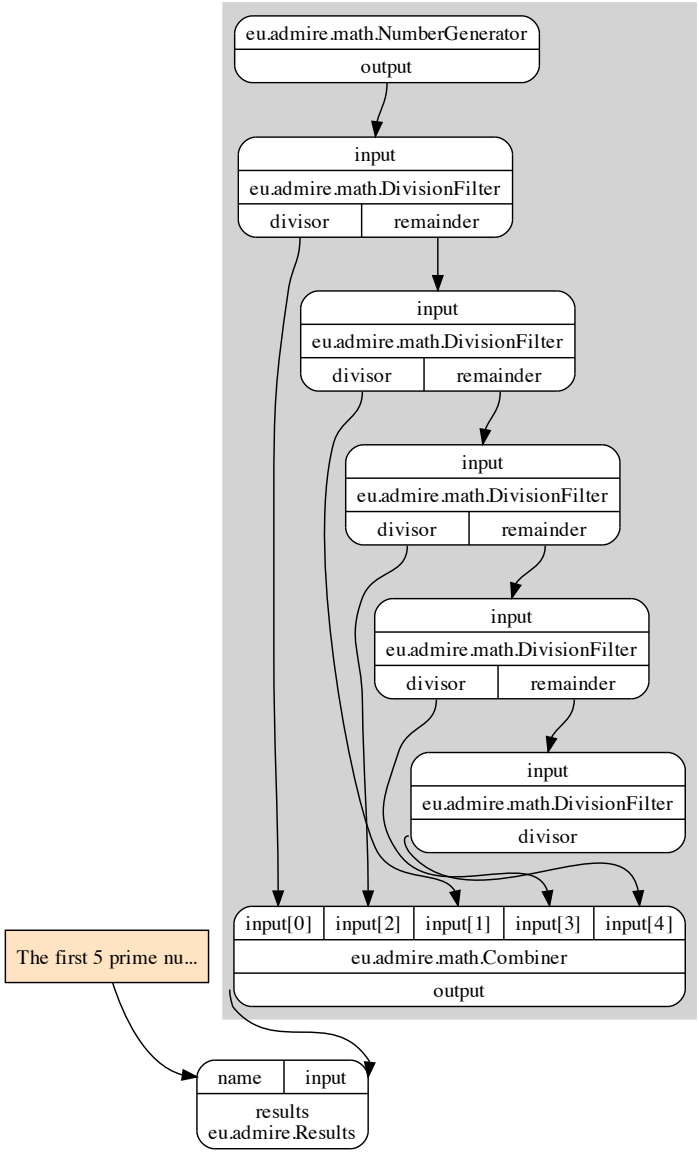
Sieve of Eratosthenes

```
1  package tutorial.example {
2    // Import sieve components and abstract type.
3    use dispel.core.Interpolate;
4    use tutorial.example.SieveFilter;
5    use dispel.math.PrimeGenerator;
6
7    // Define sieve constructor.
8    PE<PrimeGenerator> makeSieveOfEratosthenes(Integer count) {
9      // Instantiate internal components.
10     SieveFilter filter      = new SieveFilter[count];
11     Interpolate interpolate = new Interpolate with roundrobin inputs, input.length = count;
12
13     // Initialise sieve elements.
14     for (Integer i = 0; i < count - 1; i++) filter[i] = new SieveFilter with terminator output;
15     filter[count - 1] = new SieveFilter with terminator prime;
16
17     // Construct internal workflow.
18     |-x for x in 2..-| => filter[0].input;
19     for (Integer i = 0; i < count - 1; i++) {
20       filter[i].unfiltered => filter[i + 1].input;
21       filter[i].filtered   => interpolate.inputs[i];
22     }
23     filter[count - 1].unfiltered => discard;
24     filter[count - 1].filtered   => interpolate.input[count - 1];
25     filter[count - 1].filtered   => stop;
26
27     // Return all primes generated.
28     return PE( <Connection input = numbers> => <Connection primes = interpolate.output> );
29   }
30
31   // Register constructor.
32   register makeSieveOfEratosthenes;
33 }
```

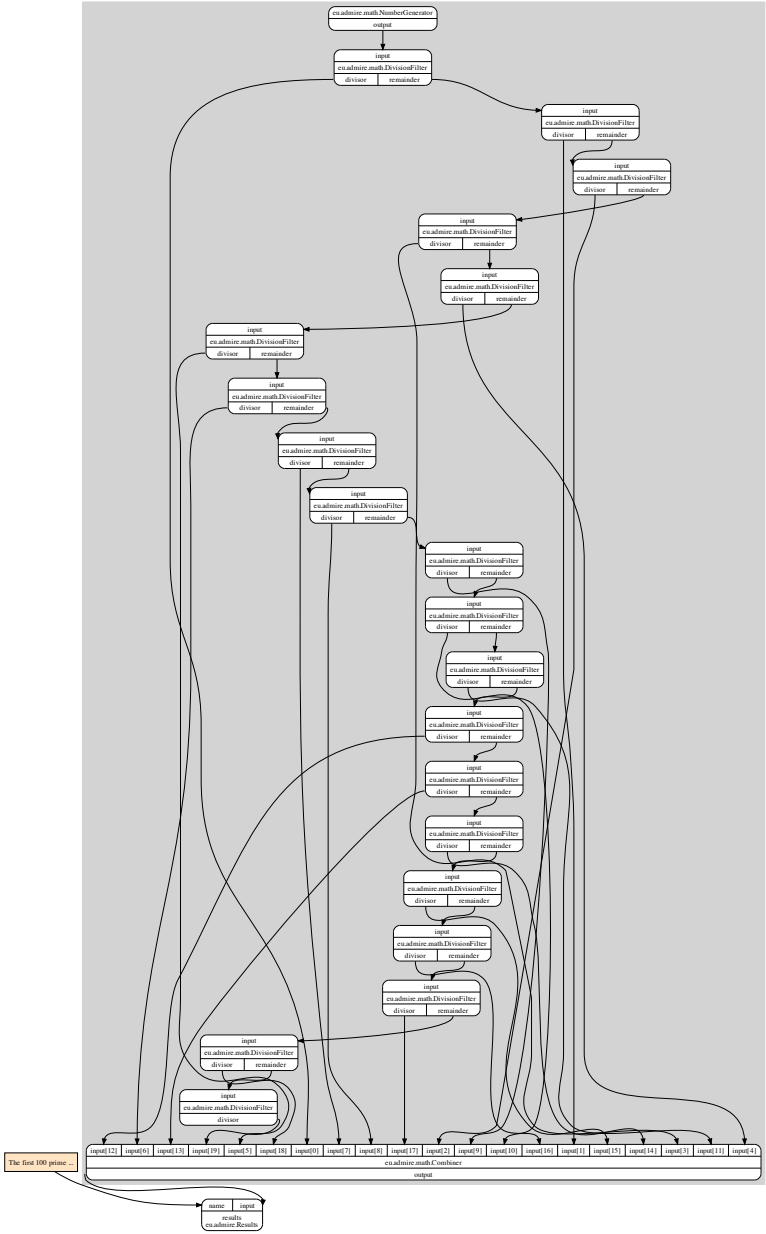
Sieve of Eratosthenes

```
1  package tutorial.example {
2    // Import abstract PE type and constructor.
3    use dispel.math.PrimeGenerator;
4    use tutorial.example.makeSieveOfEratosthenes;
5
6    // Construct the sieve.
7    PE <PrimeGenerator> SoE100 = makeSieveOfEratosthenes(100);
8    SoE100  sieve100 = new SoE100;
9    Results results  = new Results;
10
11   // Construct the top-level workflow.
12   |-"100 prime numbers"-| => results.name;
13   sieve100.primes         => results.input;
14
15   // Submit the workflow.
16   submit results;
17 }
```

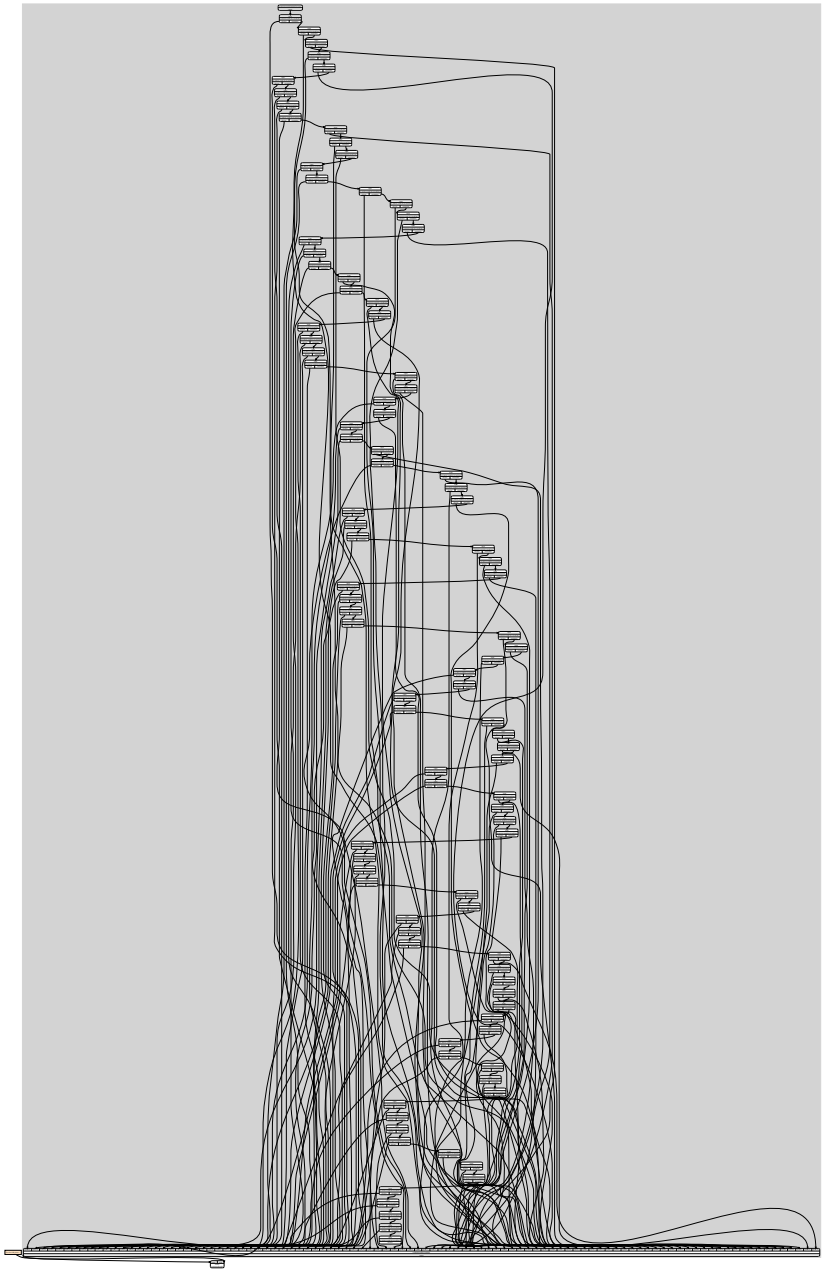
5-element sieve



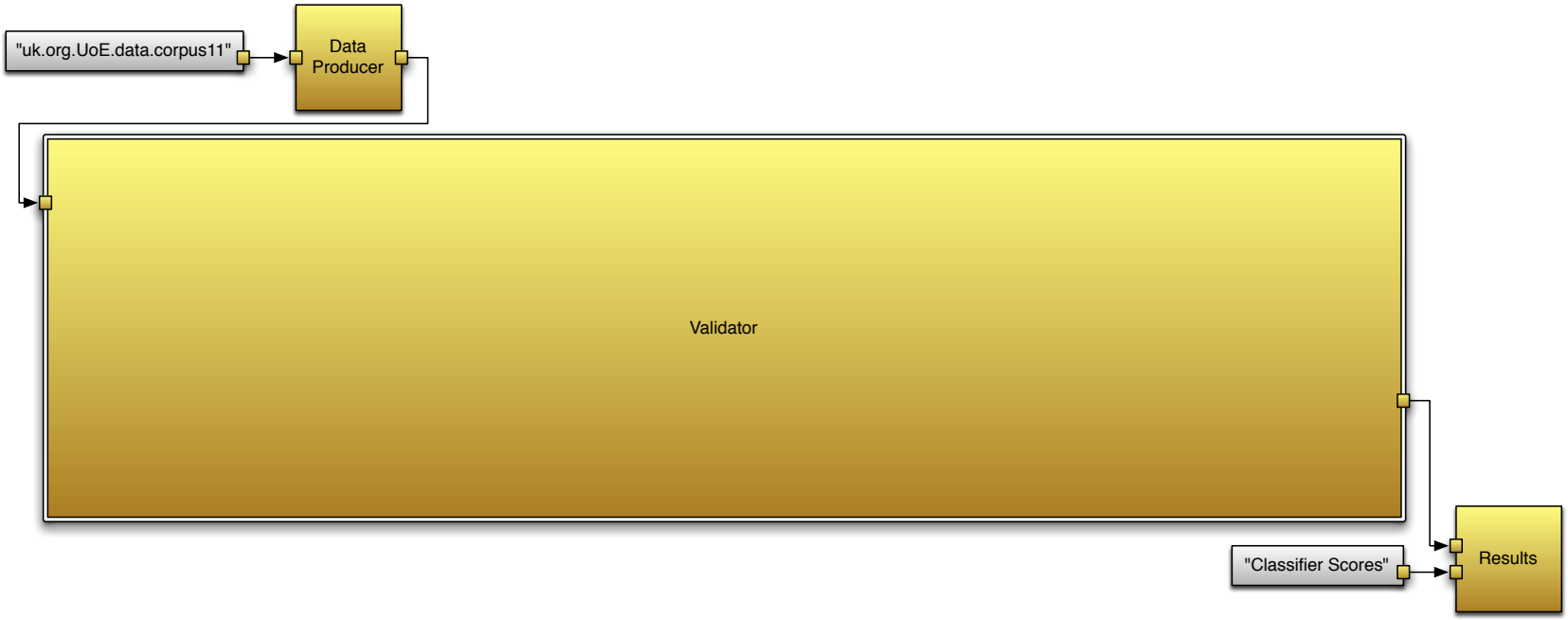
20-element sieve



100-element sieve



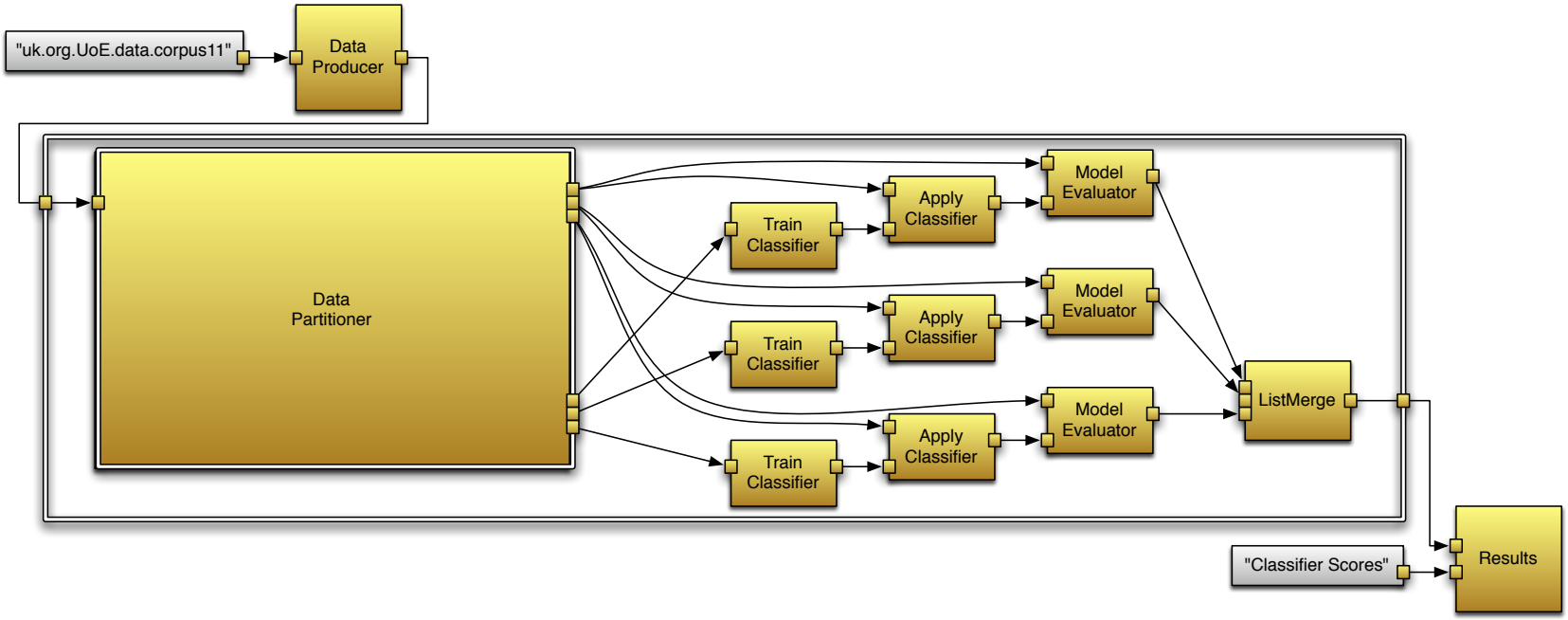
k-fold cross validation



k-fold cross validation

```
package dispel.datamining {
  // Import abstract types.
  use dispel.datamining.Validator;
  use dispel.datamining.TrainClassifier;
  use dispel.datamining.ModelEvaluator;
  use dispel.core.DataPartitioner;
  // Import PE constructor function.
  use dispel.datamining.makeDataFold;
  // Import implemented type.
  use dispel.core.ListMerge;

  // Produces a k-fold cross validation workflow pattern.
  PE<Validator> makeCrossValidator(Integer k,
                                   PE<TrainClassifier> Trainer,
                                   PE<ApplyClassifier> Classifier,
                                   PE<ModelEvaluator>  Evaluator) {
    Connection    input;
    // Data must be partitioned and re-combined for each fold.
    PE<DataPartitioner> FoldData = makeDataFold(k);
    FoldData            folder   = new FoldData;
    ListMerge           union    = new ListMerge with inputs.length = k;

    // For each fold, train a classifier then evaluate it.
    input => folder.data;
    for (Integer i = 0; i < k; i++) {
      Trainer    train     = new Trainer;
      Classifier classify  = new Classifier;
      Evaluator  evaluator = new Evaluator;

      folder.training[i] => train.data;
        train.classifier => classify.classifier;
          folder.test[i] => classify.data;
         classify.result => evaluator.predicted;
          folder.test[i] => evaluator.expected;
        evaluator.score => union.inputs[i];
    }

    // Return cross validation pattern.
    return PE( <Connection data = input> => <Connection results = union.output> );
  }

  // Register PE pattern generator.
  register makeCrossValidator;
}
```

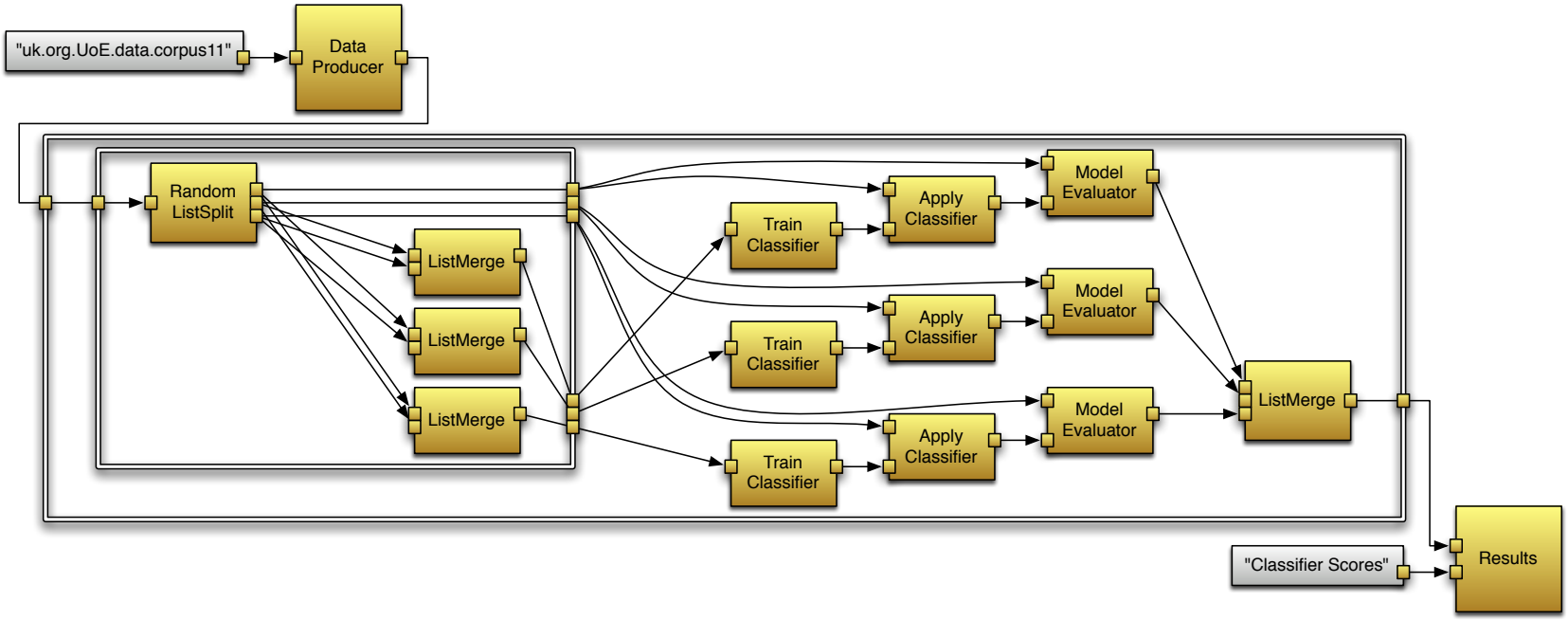
k-fold cross validation



k-fold cross validation

```
1  package dispel.datamining {
2    // Import implememed types.
3    use dispel.core.RandomListSplit;
4    use dispel.core.ListMerge;
5
6    // Produces a PE capable of splitting data for k-fold cross validation.
7    PE<DataPartitioner> makeDataFold(Integer k) {
8      Connection   input;
9      Connection[] trainingData = new Connection[k];
10     Connection[] testData     = new Connection[k];
11     // Create instance of PEs for randomly splitting and recombining data.
12     RandomListSplit sample = new RandomListSplit with results.length = k;
13     ListMerge[]     union  = new ListMerge[k];
14
15     // After partitioning data, form training and test sets.
16     input => sample.input;
17     for (Integer i = 0; i < k; i++) {
18       union[i] = new TupleUnionAll with inputs.length = k - 1;
19       for (Integer j = 0; j < i; j++) {
20         sample.outputs[j] => union[i].inputs[j];
21       }
22       sample.outputs[i] => testData[i];
23       for (Integer j = i + 1; j < k; j++) {
24         sample.outputs[j] => union[i].inputs[j - 1];
25       }
26       union[i].output => trainingData[i];
27     }
28
29     // Return data folding pattern.
30     return PE( <Connection data       = input> =>
31                <Connection[] training = trainingData;
32                 Connection[] test     = testData> );
33   }
34
35   // Register PE pattern generator.
36   register makeDataFold;
37 }
```

k-fold cross validation



k-fold cross validation

```
1  package book.examples.kdd {
2    // Import existing PEs.
3    use book.examples.kdd.DataProducer;
4    use book.examples.kdd.TrainingAlgorithmA;
5    use book.examples.kdd.BasicClassifier;
6    use book.examples.kdd.MeanEvaluator;
7    // Import abstract type and constructor.
8    use dispel.datamining.Validator;
9    use dispel.datamining.makeCrossValidator;
10
11   // Create a cross validator PE.
12   PE<Validator> CrossValidator
13     = makeCrossValidator(12, TrainingAlgorithmA, BasicClassifier, MeanEvaluator);
14   // Make instances of PEs for workflows.
15   DataProducer   producer  = new DataProducer;
16   CrossValidator validator = new CrossValidator;
17   Results        results   = new Results;
18
19   // Connect workflow.
20   |- "uk.org.UoE.data.corpus11" -| => producer.source;
21                       producer.data => validator.data;
22                  validator.results => results.input;
23           |- "Classifier Scores" -| => results.name;
24
25   // Submit workflow.
26   submit results;
27 }
```

k-fold cross validation