Introducing DISPEL

Data-Intensive Process Engineering Language

Malcolm Atkinson

DIR Theme Workshop

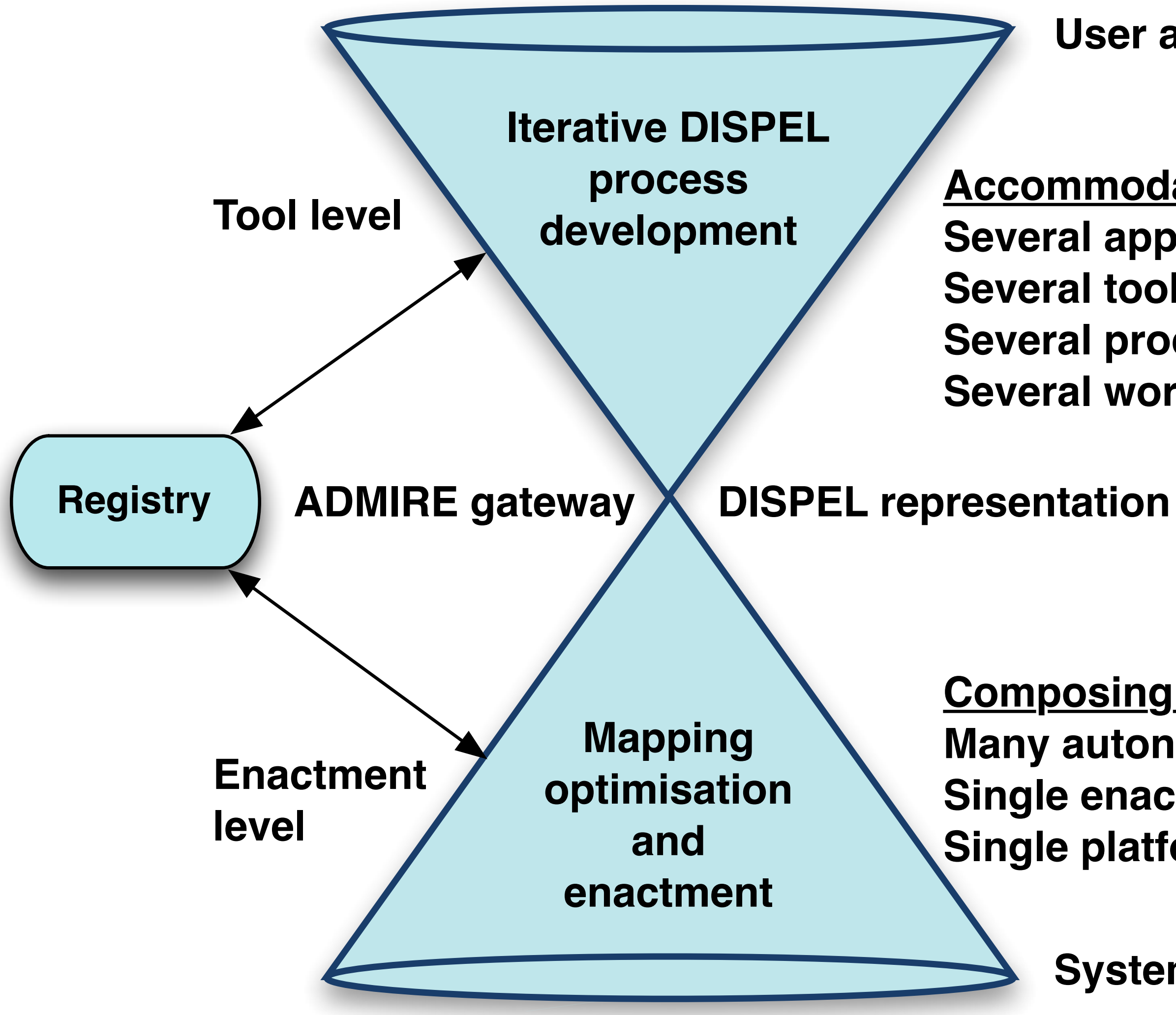
16 February 2011

www.admire-project.eu

ADMIRE – Framework 7 ICT 215024

Agenda

- DISPEL's raison d'être
- Success criteria
- Operational model
- Universe of Discourse
- Types and Validation
- Description & Context-driven graph logic



User and application diversity

<u>Accommodating</u>
Several application domains
Several tool sets
Several process representations
Several working practices

<u>Composing or hiding</u>
Many autonomous resources & services
Single enactment mechanism
Single platform implementations

System complexity

	Architectural Level		
	Tool	Gateway & DISPEL	Enactment
Domain Experts			
Data-Analysis Experts			
Data-Intensive Engineers			

DISPEL's raison d'être

- Support each category of expert
- Encourage distributed sharing
- Facilitate independent autonomous development
- Enable safe distributed evaluation
- Usual language goals
  - balance parsimony with power
  - look familiar so gets adopted

Success criteria

- To meet those goals well
  - simple composition + parametric use for Domain Experts
  - sophisticated development of algorithms and workflows for Data-Analysis Experts
  - sophisticated analysis and development of patterns for Data-Intensive Engineers
- Readable and comprehensible
- Efficient and optimisable enactment
- Integrating technical dialogue in ADMIRE

Operational model

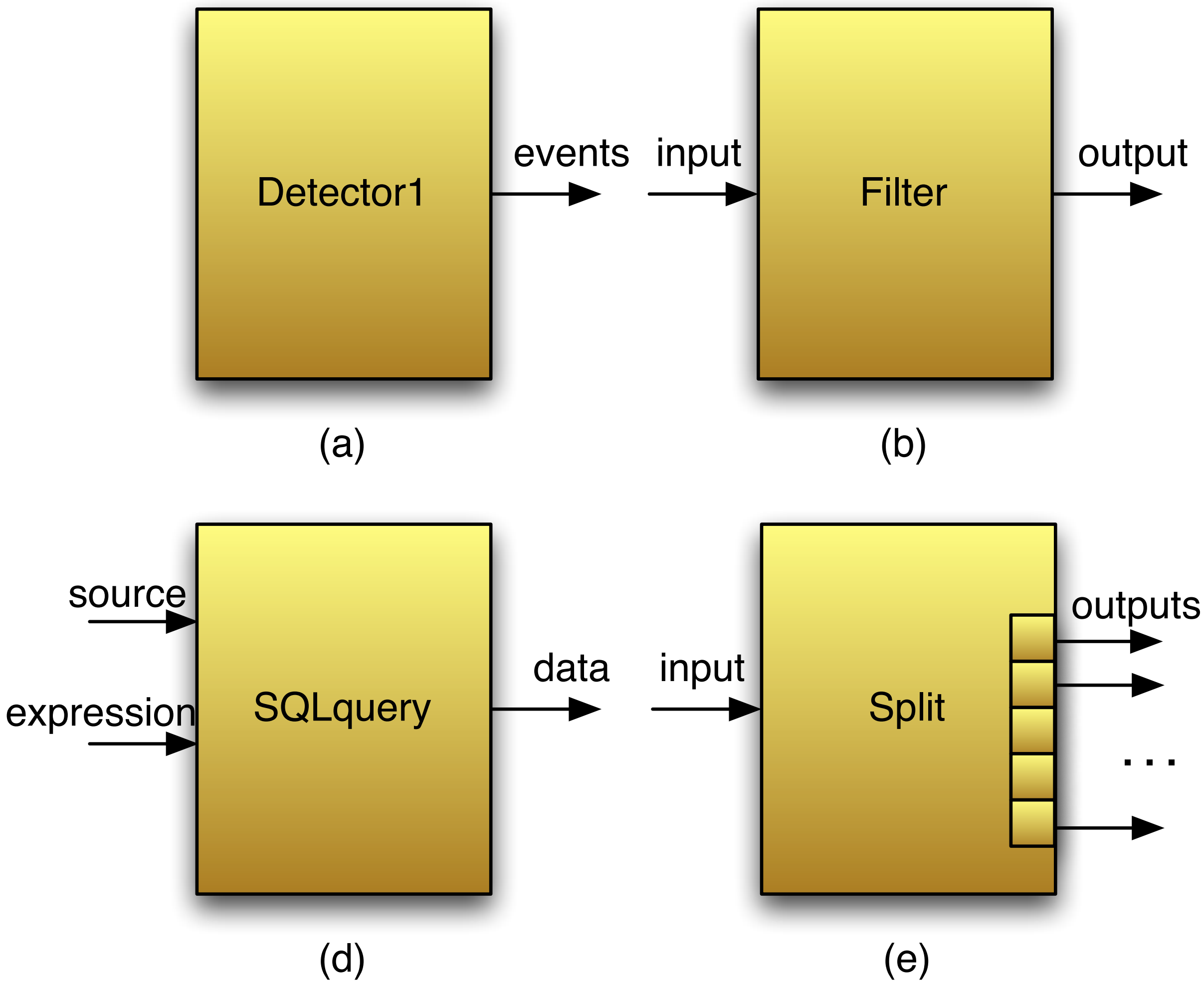
- Language processing
  - extraction from Registry
  - compilation & execution generating directed graph
- Distributed streaming enactment
  - Graph validation & optimisation
  - Resource allocation and deployment
  - Evaluation, monitoring, termination & clean up

Universe of Discourse

- Processing Elements
- Data-Streaming Connections
- Streams
- Three-level Type System
- Functions
- Registry-held descriptions
  – for all of the above
  – organised as packages/libraries

...making knowledge discovery easier

Example Processing Elements



Processing Elements

Processing Elements

- User-defined functions
  - encapsulating a data transforming algorithm

...making data-mining easier

Processing Elements

- User-defined functions
  - encapsulating a data transforming algorithm
- Expect a rich & well-organised library of PEs

Processing Elements

- User-defined functions
  - encapsulating a data transforming algorithm
- Expect a rich & well-organised library of PEs
- Expect tools that *make it easy* to
  - wrap an algorithm as a PE
  - organise a library of PEs
  - use a PE

...making data-mining easier

Processing Elements

- User-defined functions
  - encapsulating a data transforming algorithm
- Expect a rich & well-organised library of PEs
- Expect tools that *make it easy* to
  - wrap an algorithm as a PE
  - organise a library of PEs
  - use a PE
- PEs *and libraries of PEs* are
  - described in *the* registry
  - stored in the repository

...making data-mining easier

Processing Elements

- User-defined functions
  - encapsulating a data transforming algorithm
- Expect a rich & well-organised library of PEs
- Expect tools that *make it easy* to
  - wrap an algorithm as a PE
  - organise a library of PEs
  - use a PE
- PEs *and libraries of PEs* are
  - described in *the* registry
  - stored in the repository

This requires effort and a framework that we have not yet allocated

...making data-mining easier

Processing Elements

- User-defined functions
  - encapsulating a data transforming algorithm
- Expect a rich & well-organised library of PEs
- Expect tools that *make it easy* to
  - wrap an algorithm as a PE
  - organise a library of PEs
  - use a PE
- PEs *and libraries of PEs* are
  - described in *the* registry
  - stored in the repository

This requires effort and a framework that we have not yet allocated

This requires effort and a framework that we have not yet allocated

...making data-mining easier

Processing Elements have

...making data-mining easier

Processing Elements have

- A unique name

...making data-mining easier

Processing Elements have

- A unique name
- A short definition of what they do

Processing Elements have

- A unique name
- A short definition of what they do
- A precise description of their input and output streams
  - a structure of **Connections**

Processing Elements have

- A unique name
- A short definition of what they do
- A precise description of their input and output streams
  - a structure of **Connections**
- A precise description of their iterative behaviour

Processing Elements have

- A unique name
- A short definition of what they do
- A precise description of their input and output streams
  - a structure of **Connections**
- A precise description of their iterative behaviour
- A precise description of their termination and error reporting

Processing Elements have

- A unique name
- A short definition of what they do
- A precise description of their input and output streams
  - a structure of **Connections**
- A precise description of their iterative behaviour
- A precise description of their termination and error reporting
- The (S&D)type propagation rules from inputs to outputs

...making data-mining easier

Processing Elements have

- A unique name
- A short definition of what they do
- A precise description of their input and output streams
  - a structure of **Connections**
- A precise description of their iterative behaviour
- A precise description of their termination and error reporting
- The (S&D)type propagation rules from inputs to outputs
- A precise description of their properties that may permit or limit optimisation

Processing Elements have

- A unique name
- A short definition of what they do
- A precise description of their input and output streams
  - a structure of **Connections**
- A precise description of their iterative behaviour
- A precise description of their termination and error reporting
- The (S&D)type propagation rules from inputs to outputs
- A precise description of their properties that may permit or limit optimisation
- Their known subtype hierarchy

PE instances (PEIs)

- PEs are instantiated before they are used in an enactment
  - **new** PE_expression
- There may be many instances of a given PE
  - Think PE is a class
  - PEI is an instance of that class
- Assertions may refine the properties of a PE instance
  - **new** SQLquery **with** data **as** :[<Integer i, j; Real r; String s>]

...making data-mining easier

PE instances (PEIs

Stating the structural type of this particular instance's result; the programmer knows the query and schema it will be used with.

- PEs are instantiated before they are us enactment
  - **new** PE_expression
- There may be many instances of a given PE
  - Think PE is a class
  - PEI is an instance of that class
- Assertions may refine the properties of a PE instance
  - **new** SQLquery **with** data **as** :[<Integer i, j; Real r; String s>]

PE instances (PEIs)

Stating the structural type of this particular instance's result; the programmer knows the query and schema it will be used with.

- PEs are instantiated before they are used in enactment
  - **new** PE_expression
- There may be many instances of a given PE
  - Think PE is a class
  - PEI is an instance of that class
- Assertions may refine the properties of a PE instance
  - **new** SQLquery **with** data **as** :[<Integer i, j; Real r; String s>]

...making data-mining easier

Even more useful for asserting domain types

Connections

...making data-mining easier

Connections

- Connections carry a stream of data
  - from a PE instance to a PE instance
  - 1 source => multiple receivers

Connections

- Connections carry a stream of data
  - from a PE instance to a PE instance
  - 1 source => multiple receivers
- Typically a PE processes one element of the stream at a time

...making data-mining easier

Connections

- Connections carry a stream of data
  - from a PE instance to a PE instance
  - 1 source => multiple receivers
- Typically a PE processes one element of the stream at a time
- These elements are as small as makes computational sense
  - a tuple
  - a row of a matrix

Connections

- Connections carry a stream of data
  - from a PE instance to a PE instance
  - 1 source => multiple receivers
- Typically a PE processes one element of the stream at a time
- These elements are as small as makes computational sense
  - a tuple
  - a row of a matrix
- The system is responsible for buffering and optimising their flow
  - pass by reference when possible
  - serialised and compressed for long-haul data movement
  - only buffer to disk when requested or buffer spill unavoidable

Connections

- Connections carry a stream of data
  - from a PE instance to a PE instance
  - 1 source => multiple receivers
- Typically a PE processes one element of the stream at a time
- These elements are as small as makes computational sense
  - a tuple
  - a row of a matrix
- The system is responsible for buffering and optimising their flow
  - pass by reference when possible
  - serialised and compressed for long-haul data movement
  - only buffer to disk when requested or buffer spill unavoidable

The "make a connection" operator

...making data-mining easier

Connections

...making data-mining easier

Connections

- Two types describe the values passed
  - structural type (**Stype**)
    - the format / representation of the elemental value
  - domain type (**Dtype**)
    - the `meaning' of the elemental value

Connections

- Two types describe the values passed
  - structural type (**Stype**)
    - the format / representation of the elemental value
  - domain type (**Dtype**)
    - the `meaning' of the elemental value
- Connections may have finite or continuous streams
  - Stream end, **EoS,** indicates no more data available
    - A PE transmits **EoS** when it has no more data to send
  - A connection may transmit a "no more" message from receiver to source

Connections

- Two types describe the values passed
  - structural type (**Stype**)
    - the format / representation of the elemental value
  - domain type (**Dtype**)
    - the `meaning' of the elemental value
- Connections may have finite or continuous streams
  - Stream end, **EoS,** indicates no more data available
    - A PE transmits **EoS** when it has no more data to send
  - A connection may transmit a "no more" message from receiver to source
- Receiver **discard** throws away data
  - it sends a "no more" message immediately

Connections

- Two types describe the values passed
  - structural type (**Stype**)
    - the format / representation of the elemental value
  - domain type (**Dtype**)
    - the `meaning' of the elemental value
- Connections may have finite or continuous streams
  - Stream end, **EoS,** indicates no more data available
    - A PE transmits **EoS** when it has no more data to send
  - A connection may transmit a "no more" message from receiver to source
- Receiver **discard** throws away data
  - it sends a "no more" message immediately
- Stream literals have the form
  - |- expression -|

...making data-mining easier

Connections

- Two types describe the values passed
  - structural type (**Stype**)
    - the format / representation of the elemental value
  - domain type (**Dtype**)
    - the `meaning’ of the elemental value
- Connections may have finite or continuous streams
  - Stream end, **EoS,** indicates no more data available
    - A PE transmits **EoS** when it has no more data to send
  - A connection may transmit a “no more” message from receiver to source
- Receiver **discard** throws away data
  - it sends a “no more” message immediately
- Stream literals have the form
  - |- expression -|

keywords for declaring structural and domain types

Start and end of stream symbols

...making data-mining easier

PE Termination

...making data-mining easier

PE Termination

- The *default* termination behaviour occurs when either all the inputs are exhausted or all the receivers of outputs have indicated they do not want more data
  - When all of a PE's inputs have received **EoS**
    - a PE completes the use of its current data
    - then sends an EoS on all of its outputs
    - then stops
  - When all of a PE's outputs have received a "no more"
    - a PE sends a "no more" on all of its inputs
    - then stops

PE Termination

- The *default* termination behaviour occurs when either all the inputs are exhausted or all the receivers of outputs have indicated they do not want more data
  - When all of a PE's inputs have received **EoS**
    - a PE completes the use of its current data
    - then sends an EoS on all of its outputs
    - then stops
  - When all of a PE's outputs have received a "no more"
    - a PE sends a "no more" on all of its inputs
    - then stops
- Termination should propagate across a distributed enactment
  - Receipt of **stop** event as well

PE Termination

- The *default* termination behaviour occurs when either all the inputs are exhausted or all the receivers of outputs have indicated they do not want more data
  - When all of a PE's inputs have received **EoS**
    - a PE completes the use of its current data
    - then sends an EoS on all of its outputs
    - then stops
  - When all of a PE's outputs have received a "no more"
    - a PE sends a "no more" on all of its inputs
    - then stops
- Termination should propagate across a distributed enactment
  - Receipt of **stop** event as well

This is the default, a PE may stop when a particular stream delivers EoS

...making data-mining easier

PE Termination

- The *default* termination behaviour occurs when either all the inputs are exhausted or all the receivers of outputs have indicated they do not want more data
  - When all of a PE's inputs have received **EoS**
    - a PE completes the use of its current data
    - then sends an EoS on all of its outputs
    - then stops
  - When all of a PE's outputs have received a "no more"
    - a PE sends a "no more" on all of its inputs
    - then stops
- Termination should propagate across a distributed
  - Receipt of **stop** event as well

This is the default, a PE may stop when a particular stream delivers EoS

This is the default, a PE may stop when a particular stream receives "no more"

...making d

PE Termination

- The *default* termination behaviour occurs when either all the inputs are exhausted or all the receivers of outputs have indicated they do not want more data
  - When all of a PE's inputs have received **EoS**
    - a PE completes the use of its current data
    - then sends an EoS on all of its outputs
    - then stops
  - When all of a PE's outputs have received a "no more"
    - a PE sends a "no more" on all of its inputs
    - then stops
- Termination should propagate across a distributed
  - Receipt of **stop** event as well

This is the default, a PE may stop when a particular stream delivers EoS

This is the default, a PE may stop when a particular stream receives "no more"

...making d

But see later slides on PE type definitions and assertions for **terminator** keyword

Types and Validation

- Language Types
  - ensuring correct composition of terms in DISPEL
- Structural Types
  - tracking consistent structure/format along connections
  - PE input requirements output assertions
- Domain Types
  - tracking consistent meaning / interpretation along connections
  - PE input requirements and output assertions

A PE with specific termination

```
package eu.admire{
  namespace rdb “http://www.iso.org/iso/concept_database_cdb#”;

  Type SQLquery is PE (
    <Connection locator: String:: “rdb:Database_URI” resource;
      Connection terminator: String:: “rdb:SQL_Query_Statement” expression > =>
    <Connection: [<rest>]:: “rdb:Relational_Result_Set” data> );

  register SQLquery;
}
```

A PE with specific termination

```
package eu.admire{
  namespace rdb "http://www.iso.org/iso/concept_database_cdb#";

  Type SQLquery is PE (
    <Connection locator: String:: "rdb:Database_URI" resource;
     Connection terminator: String:: "rdb:SQL_Query_Statement" expression > =>
    <Connection: [<rest>]:: "rdb:Relational_Result_Set" data> );

  register SQLquery;
}
```

Declare a PE type that takes a URI to specify the RDB to which the query supplied by expression is sent.

A PE with specific termination

Indicate that no more data on this input *alone* causes this PE to terminate

```
package eu.admire{
  namespace rdb “http://www.iso.org/iso/concept_database_cdb#”;

Type SQLquery is PE (
   <Connection locator: String:: “rdb:Database_URI” resource;
     Connection terminator: String:: “rdb:SQL_Query_Statement” expression > =>
   <Connection: [<rest>]:: “rdb:Relational_Result_Set” data> );

  register SQLquery;
}
```

Declare a PE type that takes a URI to specify the RDB to which the query supplied by expression is sent.

A PE with specific termination

Indicate that no more data on this input *alone* causes this PE to terminate

```
package eu.admire{
  namespace rdb "http://www.iso.org/iso/concept_database_cdb#";

  Type SQLquery is PE (
    <Connection locator: String:: "rdb:Database_URI" resource;
     Connection terminator: String:: "rdb:SQL_Query_Statement" expression > =>
    <Connection: [<rest>]:: "rdb:Relational_Result_Set" data> );

  register SQLquery;
}
```

The structural type of the result is a list of tuples; **rest** indicates we don't know anything about the fields in the tuples

Declare a PE type that takes a URI to specify the RDB to which the query supplied by expression is sent.

A PE with specific termination

make rdb an abbreviation for the domain name prefix

```
package eu.admire{
 namespace rdb “http://www.iso.org/iso/concept_database_cdb#”;

 Type SQLquery is PE (
   <Connection locator: String:: “rdb:Database_URI” resource;
     Connection terminator: String:: “rdb:SQL_Query_Statement” expression > =>
   <Connection: [<rest>]:: “rdb:Relational_Result_Set” data> );

 register SQLquery;
}
```

A PE with specific termination

make rdb an abbreviation for the domain name prefix

package eu.admire{
namespace rdb "http://www.iso.org/iso/concept_database_cdb#";

Type SQLquery **is PE** (
<**Connection locator**: String:: "rdb:Database_URI" resource;
Connection terminator: String:: "rdb:SQL_Query_Statement" expression > =>
<**Connection**: [<**rest**>]:: "rdb:Relational_Result_Set" data>);

register SQLquery
}

The Connection attribute **locator** indicates that this a PE whose instances might usefully be `anchored' close to the specified data source by the optimiser. This is particularly relevant if this input is connected to a stream literal.

Defining a DISPEL function

```
package eu.admire{
  namespace rdb "http://www.iso.org/iso/concept_database_cdb#";
  use uk.org.ogsadai.SQLQuery;

  Type SQLQtype is PE(
    <Connection: String:: "rdb:SQL_Query_Statement" expression> =>
    <Connection: [<rest>]:: "rdb:Relational_Result_Set" data>
  );

  PE<SQLQtype> lockSQLdataSource(String dataSource) {
    SQLquery sqlq = new SQLquery;

    |- repeat enough of dataSource -|:String::"rdb:Database_URI" =>sqlq.resource;

    return SQLQtype( <Connection expression = sqlq.expression> =>
                <Connection data = sqlq.data> );
  };

  register SQLQtype, lockSQLdataSource;
}
```

Defining a DISPEL function

```
package eu.admire{
  namespace rdb “http://www.iso.org/iso/concept_database_cdb#”;
  use uk.org.ogsadai.SQLQuery;

  Type SQLQtype is PE(
    <Connection: String:: “rdb:SQL_Query_Statement” expression> =>
    <Connection: [<rest>]:: “rdb:Relational_Result_Set” data>
  );

  PE<SQLQtype> lockSQLdataSource(String dataSource) {
    SQLquery sqlq = new SQLquery;

    |- repeat enough of dataSource -|:String::“rdb:Database_URI” =>sqlq.resource;

    return SQLQtype( <Connection expression = sqlq.expression> =>
                     <Connection data = sqlq.data> );
  };

  register SQLQtype, lockSQLdataSource;
}
```

Defining the result type

Defining a DISPEL function

```
package eu.admire{
  namespace rdb "http://www.iso.org/iso/concept_database_cdb#";
  use uk.org.ogsadai.SQLQuery;

  Type SQLQtype is PE(
    <Connection: String:: "rdb:SQL_Query_Statement" expression> =>
    <Connection: [<rest>]:: "rdb:Relational_Result_Set" data>
  );

  PE<SQLQtype> lockSQLdataSource(String dataSource) {
    SQLquery sqlq = new SQLquery;

    |- repeat enough of dataSource -|:String::"rdb:Database_URI" =>sqlq.resource;

    return SQLQtype( <Connection expression = sqlq.expression> =>
                <Connection data = sqlq.data> );
  };

  register SQLQtype, lockSQLdataSource;
}
```

Defining the result type

Defining the function

Defining a DISPEL function

```
package eu.admire{
  namespace rdb "http://www.iso.org/iso/concept_database_cdb#";
  use uk.org.ogsadai.SQLQuery;

  Type SQLQtype is PE(
    <Connection: String:: "rdb:SQL_Query_Statement" expression> =>
    <Connection: [<rest>]:: "rdb:Relational_Result_Set" data>
  );

  PE<SQLQtype> lockSQLdataSource(String dataSource) {
    SQLquery sqlq = new SQLquery;

    |- repeat enough of dataSource -|:String::"rdb:Database_URI" =>sqlq.resource;

    return SQLQtype( <Connection expression = sqlq.expression> =>
                <Connection data = sqlq.data> );
  };

  register SQLQtype, lockSQLdataSource;
}
```

The result type is a sub-type of SQLQ-type

Defining the function

Defining a DISPEL function

```
package eu.admire{
  namespace rdb "http://www.iso.org/iso/concept_database_cdb#";
  use uk.org.ogsadai.SQLQuery;

  Type SQLQtype is PE(
    <Connection: String:: "rdb:SQL_Query_Statement" expression> =>
    <Connection: [<rest>]:: "rdb:Relational_Result_Set" data>
  );

  PE<SQLQtype> lockSQLdataSource(String dataSource) {
    SQLquery sqlq = new SQLquery;

    |- repeat enough of dataSource -|:String::"rdb:Database_URI" =>sqlq.resource;

    return SQLQtype( <Connection expression = sqlq.expression> =>
                     <Connection data = sqlq.data> );
  };

  register SQLQtype, lockSQLdataSource;
}
```

The result type is a sub-type of SQLQ-type

Defining the function

The parameter

Defining a DISPEL function

```
package eu.admire{
  namespace rdb "http://www.iso.org/iso/concept_database_cdb#";
  use uk.org.ogsadai.SQLQuery;

  Type SQLQtype is PE(
    <Connection: String:: "rdb:SQL_Query_Statement" expression> =>
    <Connection: [<rest>]:: "rdb:Relational_Result_Set" data>
  );

  PE<SQLQtype> lockSQLdataSource(String dataSource) {
    SQLquery sqlq = new SQLquery;

    |- repeat enough of dataSource -|:String::"rdb:Database_URI" =>sqlq.resource;

    return SQLQtype( <Connection expression = sqlq.expression> =>
        <Connection data = sqlq.data> );
  };

  register SQLQtype, lockSQLdataSource;
}
```

The result type is a sub-type of SQLQ-type

The parameter

Constructing PE result

Defining a DISPEL function

```
package eu.admire{
  namespace rdb "http://www.iso.org/iso/concept_database_cdb#";
  use uk.org.ogsadai.SQLQuery;

  Type SQLQtype is PE(
    <Connection: String:: "rdb:SQL_Query_Statement" expression> =>
    <Connection: [<rest>]:: "rdb:Relational_Result_Set" data>
  );

  PE<SQLQtype> lockSQLdataSource(String dataSource) {
    SQLquery sqlq = new SQLquery;

    |- repeat enough of dataSource -|:String::"rdb:Database_URI" =>sqlq.resource;

    return SQLQtype( <Connection expression = sqlq.expression> =>
          <Connection data = sqlq.data> );
  };

  register SQLQtype, lockSQLdataSource;
}
```

The result type is a sub-type of SQLQ-type

The parameter

Constructing PE result

Constructing an infinite sequence of the supplied URI

Defining a DISPEL function

```
package eu.admire{
  namespace rdb "http://www.iso.org/iso/concept_database_cdb#";
  use uk.org.ogsadai.SQLQuery;

  Type SQLQtype is PE(
    <Connection: String:: "rdb:SQL_Query_Statement" expression> =>
    <Connection: [<rest>]:: "rdb:Relational_Result_Set" data>
  );

  PE<SQLQtype> lockSQLdataSource(String dataSource) {
    SQLquery sqlq = new SQLquery;

    |- repeat enough of dataSource -|:String::"rdb:Database_URI" =>sqlq.resource;

    return SQLQtype( <Connection expression = sqlq.expression> =>
                     <Connection data = sqlq.data> );
  };

  register SQLQtype, lockSQLdataSource;
}
```

Make available for future **use**

Make n-way merge tree

```
package eu.admire {
  use eu.admire.Combiner;

  PE <Combiner> makeMwayCombiner(Integer m, PE <Combiner> BC) {
    if ((new BC).inputs.length != 2) then  {
      throw new ParameterException(
      "BC.inputs parameter provided to makeMwayCombiner not binary" );
      return null;
    };
    if (m <= 0) then {
      throw new ParameterException( "m <= 0 in makeMwayCombiner" );
      return null;
    };
    if (m == 1 ) then  {
      BC bc = new BC;
      |- -| => bc.inputs[1];
      return Combiner( <Connection inputs[0] = bc.inputs[0]> =>
                       <Connection output = bc.output> );
    };
    if (m == 2 ) then {
        return BC;
    };
```

Make n-way merge tree

```
package eu.admire {
  use eu.admire.Combiner;

  PE <Combiner> makeMwayCombiner(Integer m, PE <Combiner> BC) {
    if ((new BC).inputs.length != 2) then  {
      throw new ParameterException(
      "BC.inputs parameter provided to makeMwayCombiner not binary" );
      return null;
    };
    if (m <= 0) then {
      throw new ParameterException( "m <= 0 in makeMwayCombiner" );
      return null;
    };
    if (m == 1 ) then  {
      BC bc = new BC;
      |- -| => bc.inputs[1];
      return Combiner( <Connection in
                       <Connection
    };
    if (m == 2 ) then {
        return BC;
    };
```

Defensively catch errors in the supplied parameters. Normally necessary but not shown in all the other patterns.

Make n-way merge tree

Deal with special cases and bottom out the recursive build.

```
package eu.admire {
  use eu.admire.Combiner;

  PE <Combiner> makeMwayCombiner(Integer m, PE <Combiner> BC) {
    if ((new BC).inputs.length != 2) then  {
      throw new ParameterException(
      "BC.inputs parameter provided to makeMwayCombiner not binary" );
      return null;
    };
    if (m <= 0) then {
      throw new ParameterException( "m <= 0 in makeMwayCombiner" );
      return null;
    };
    if (m == 1 ) then  {
      BC bc = new BC;
      |- -| => bc.inputs[1];
      return Combiner( <Connection inputs[0] = bc.inputs[0]> =>
                       <Connection output = bc.output> );
    };
    if (m == 2 ) then {
        return BC;
    };
```

```
    return Combiner( <Connection inputs[0] = bc.inputs[0]> =>
                             <Connection output = bc.output> );
  };
  if (m == 2 ) then {
    return BC;
  };
  Connection[ ] inputs = new Connection[m];
  Integer half = m / 2;
  Integer otherHalf = m - half;
  Combiner FirstTree = makeMwayCombiner( half, BC );
  Combiner SecondTree = makeMwayCombiner( otherHalf, BC );
  Combiner ft = new FirstTree;
  Combiner st = new SecondTree;
  Combiner base = new BC;
  Integer deposit = 0;
  for (Integer i = 0; i<half; i++) {
    inputs[deposit] => ft.inputs[i];
    deposit++;
  };
  for (Integer i = 0; i<otherHalf; i++) {
    inputs[deposit] => st.inputs[i];
    deposit++;
  };
  ft.output => base.inputs[0];
  st.output => base.inputs[1];
  return Combiner( <Connection[ ] inputs = inputs> =>
                          <Connection output = base.output> );
}
```

```
    return Combiner( <Connection inputs[0] = bc.inpu
                     <Connection outp
  };
  if (m == 2 ) then {
    return BC;
  };
  Connection[ ] inputs = new Connection[m];
  Integer half = m / 2;
  Integer otherHalf = m - half;
  Combiner FirstTree = makeMwayCombiner( half, BC );
  Combiner SecondTree = makeMwayCombiner( otherHalf, BC );
  Combiner ft = new FirstTree;
  Combiner st = new SecondTree;
  Combiner base = new BC;
  Integer deposit = 0;
  for (Integer i = 0; i<half; i++) {
    inputs[deposit] => ft.inputs[i];
    deposit++;
  };
  for (Integer i = 0; i<otherHalf; i++) {
    inputs[deposit] => st.inputs[i];
    deposit++;
  };
  ft.output => base.inputs[0];
  st.output => base.inputs[1];
  return Combiner( <Connection[ ] inputs = inputs> =>
                   <Connection output = base.output> );
}
```

Build two subtrees and join them together with an instance of the supplied binary combiner BC

```
    Combiner FirstTree = makeMwayCombiner( half, BC );
    Combiner SecondTree = makeMwayCombiner( otherHalf, BC );
    Combiner ft = new FirstTree;
    Combiner st = new SecondTree;
    Combiner base = new BC;
    Integer deposit = 0;
    for (Integer i = 0; i<half; i++) {
      inputs[deposit] => ft.inputs[i];
      deposit++;
    };
    for (Integer i = 0; i<otherHalf; i++) {
      inputs[deposit] => st.inputs[i];
      deposit++;
    };
    ft.output => base.inputs[0];
    st.output => base.inputs[1];
    return Combiner( <Connection[ ] inputs = inputs> =>
                     <Connection output = base.output> );
  }

  register makeMwayCombiner;
}
```

```
    Combiner FirstTree = makeMwayCombiner( half, BC );
    Combiner SecondTree = makeMwayCombiner( otherHalf, BC );
    Combiner ft = new FirstTree;
    Combiner st = new SecondTree;
    Combiner base = new BC;
    Integer deposit = 0;
    for (Integer i = 0; i<half; i++) {
      inputs[deposit] => ft.inputs[i];
      deposit++;
    };
    for (Integer i = 0; i<otherHalf; i++) {
      inputs[deposit] => st.inputs[i];
      deposit++;
    };
    ft.output => base.inputs[0];
    st.output => base.inputs[1];
    return Combiner( <Connection[ ] inputs = inputs> =>
                     <Connection output = base.output> );
  }

  register makeMwayCombiner;
}
```

Example 29: wiki/MeetingMinutes/Architecture/DISPEL/MakeTreeOfCombiners

An example of makeMwayCombiner(4, Amerge)