Virtual Earthquake and seismology Research Community e-science environment in Europe
Project 283543 – FP7-INFRASTRUCTURES-2011-2 – www.verce.eu – info@verce.eu

DISPEL Enactment

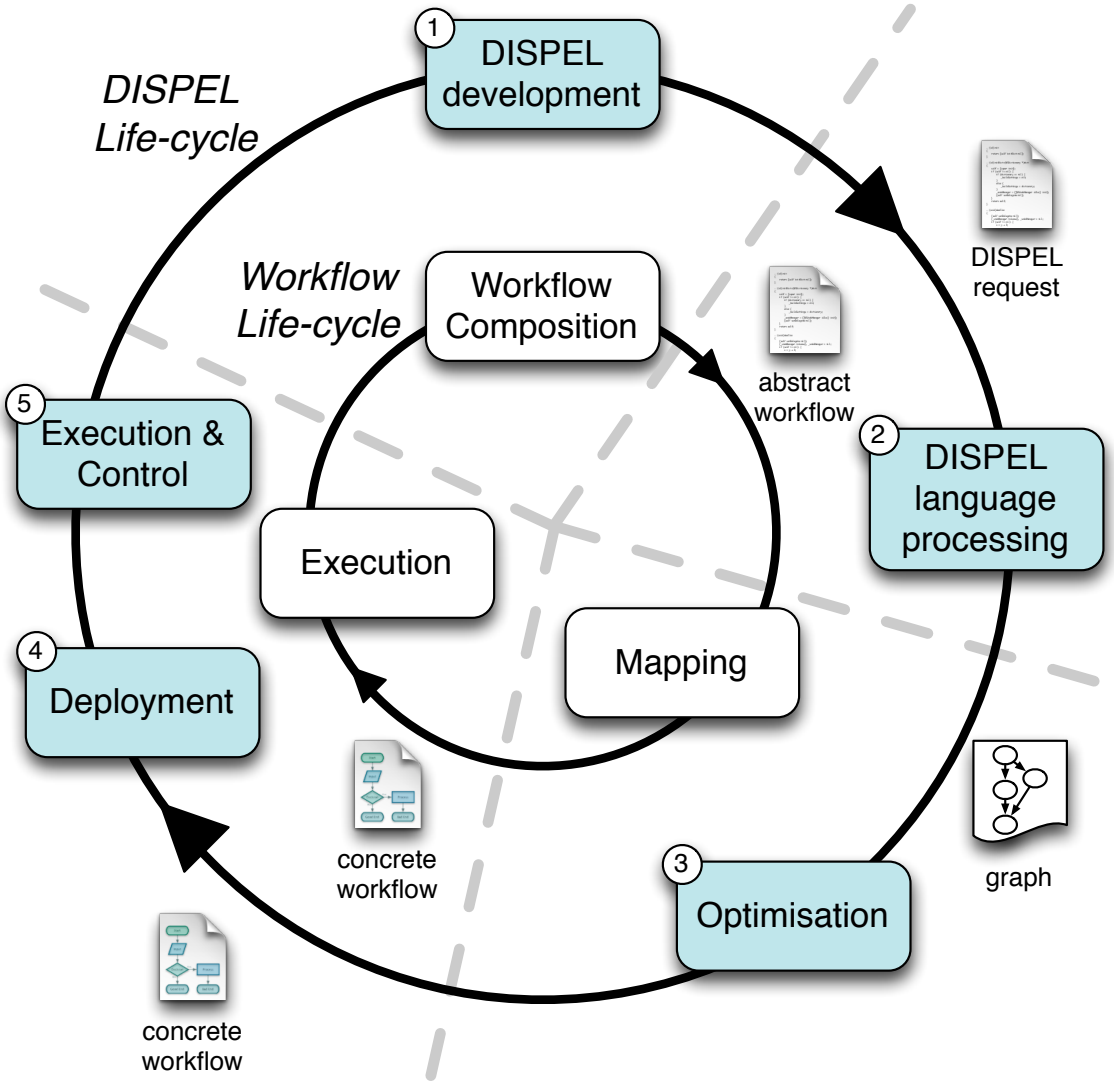
Amy Krause [1] and Chee Sun Liew [2]

[1] EPCC, University of Edinburgh

[2] Data-Intensive Research Group, University of Edinburgh

VERCE @ University of Liverpool, 3 September 2012

DISPEL life-cycle



Enactment Model

1. DISPEL language processing
   1. Validation & import from registry
   2. Format & meaning mis-match handling
   3. Interpretation to generate graph
2. Graph optimisation & mapping
   1. Re-ordering & parallelisation
   2. Identification of where to do the work
   3. Selection of PE implementations & instances
   4. Partitioning into co-located subgraphs
3. Deployment & initialisation
4. Execution, Monitoring & Clean up

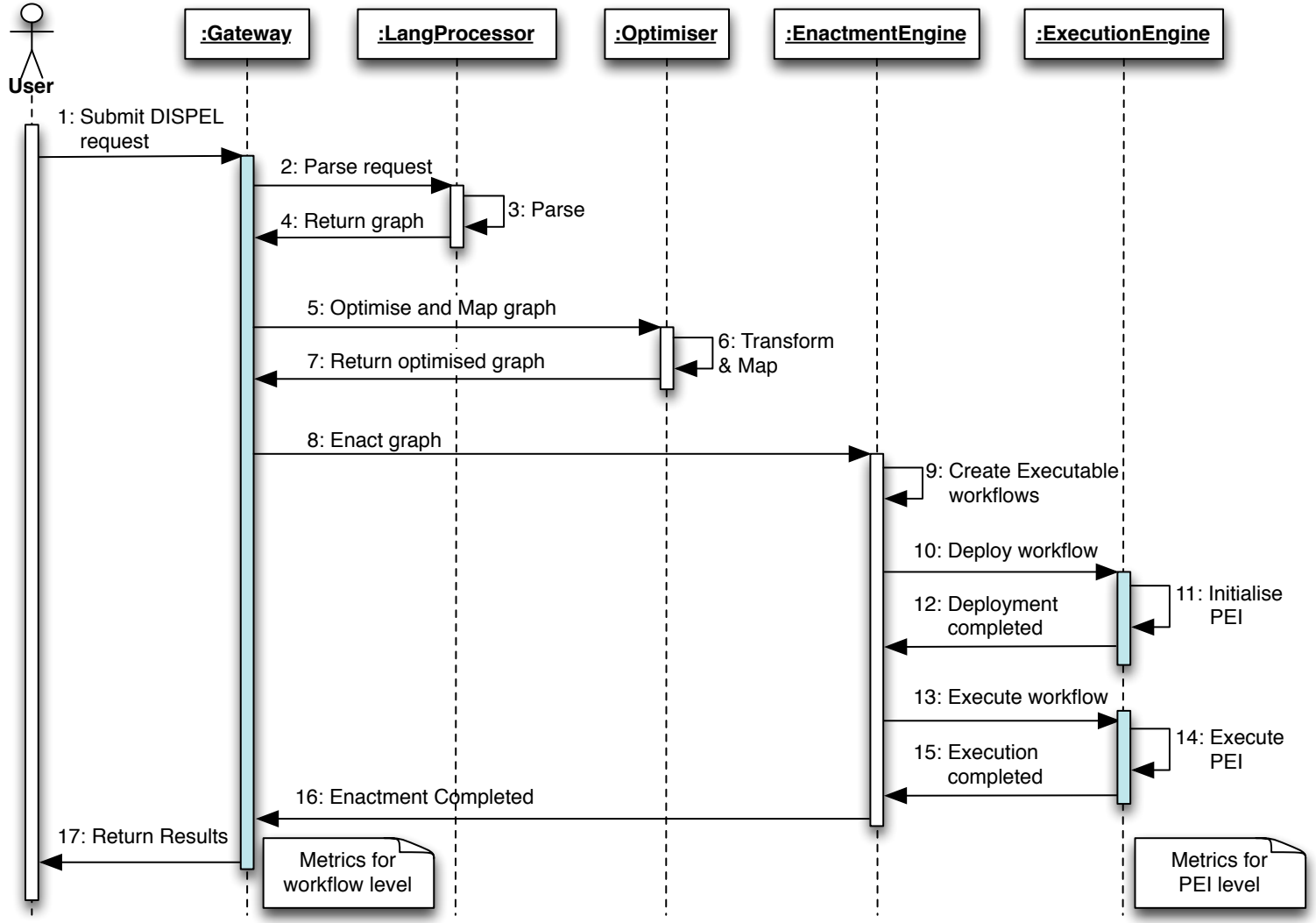
Preparing to do the data processing

Doing the data processing

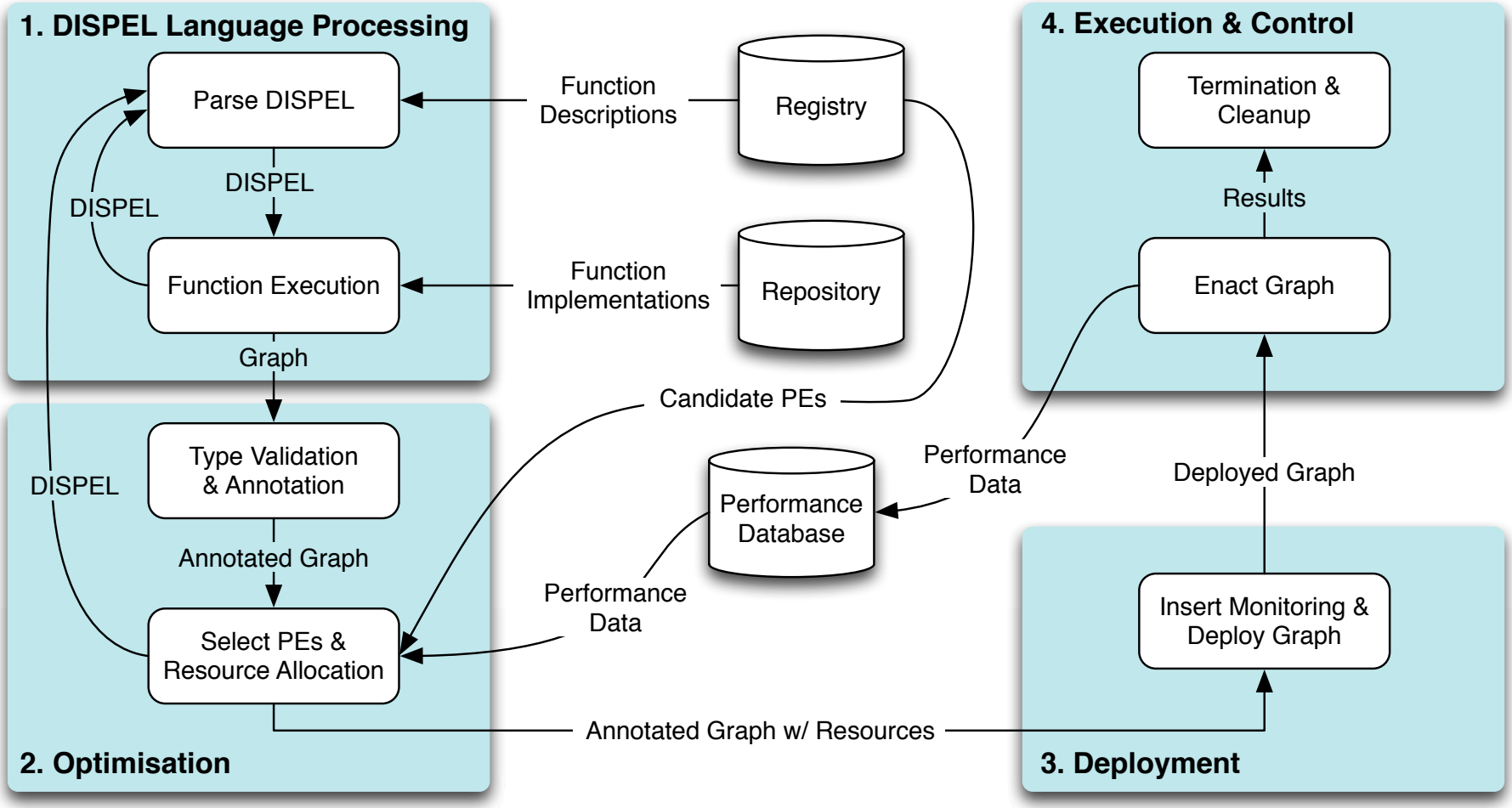
Data-Intensive platform

- an application development environment
  - (including libraries of processing elements, functions, and data types),
- a gateway
  - as the entry point of enactment which accepts DISPEL request,
- a DISPEL language processor
  - that compiles the DISPEL request into graph representation,
- an enactment engine
  - that optimises those graphs, deploys them, executes them in a controllable framework that permits interaction with the end user, and finally terminates them and cleans up the environment,
- execution engines
  - that deploy and execute workflows, and
- data sources
  - that are connected and made available through this platform.

Execution model of a DISPEL request



DISPEL enactment process



Four stages in the enactment process

- DISPEL Language Processing
  - parsing and validating a DISPEL program and creating the data-flow graphs.
- Optimisation
  - selection of PEs, transformation of the data flow graph, substitution of PEs, identification of available resources, and the mapping of PEs to resources.
- Deployment
  - translation into platform-specific form and initialising resources and connections.
- Execution and Control
  - instrumentation and performance measurement, failure management, delivering results and clean up.

Stage 1: DISPEL Language Processing

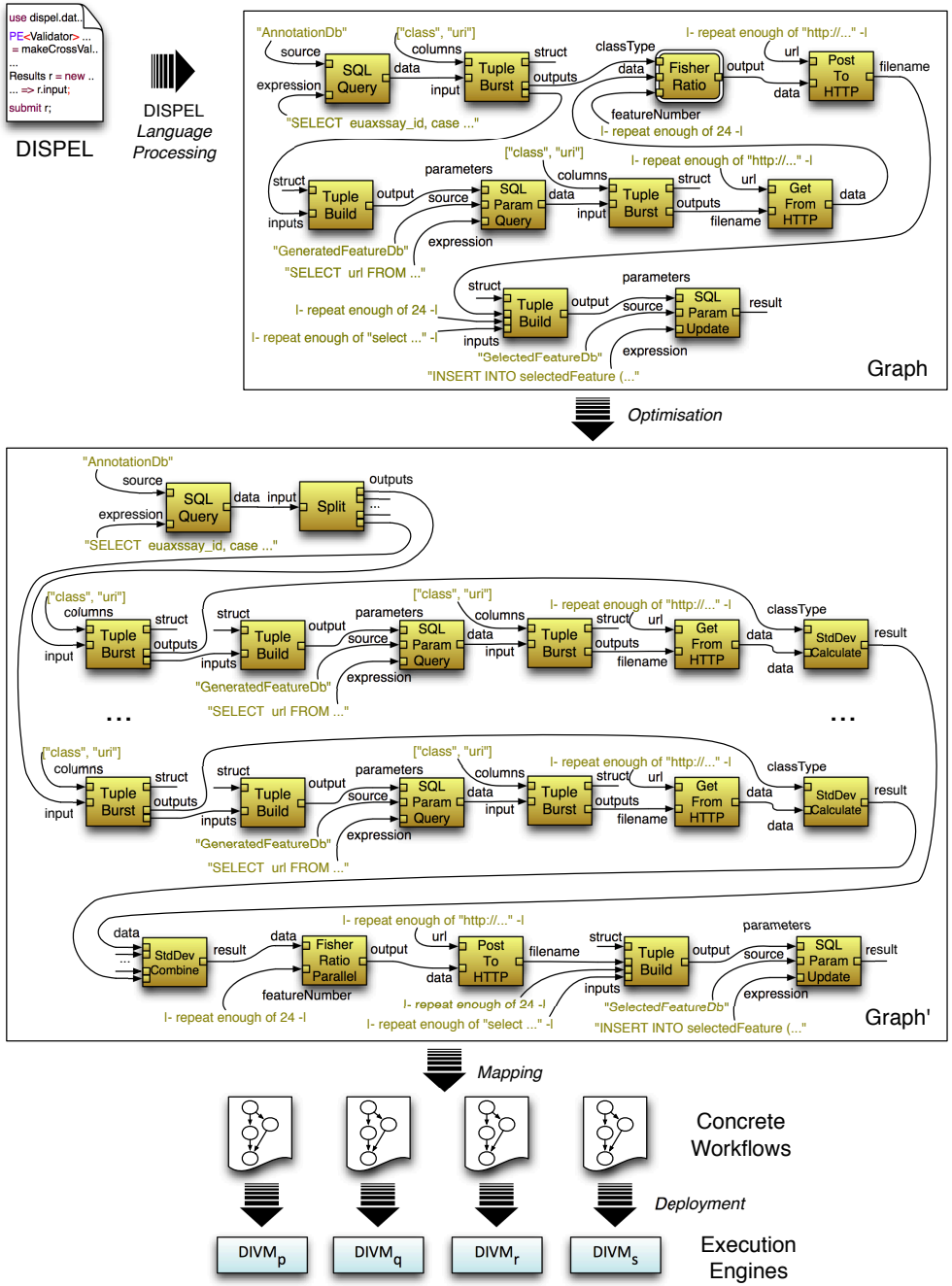
- Compilation
  - DISPEL request parsing and syntax validation
  - Import used types and objects from the registry
- DISPEL graph generation
  - Interpret the DISPEL request
  - Execute functions, statements, conditionals, loops
  - Connect processing elements to form graphs
- Registration
  - Types and objects may be registered in the ADMIRE registry for re-use

Stage 2: Optimisation

- Graph transformation
  - Sub-graph substitution
  - Parallelisation
  - Reordering
- Resource allocation - where to do the work
  - Mapping data resources to physical resources
  - Partitioning the graph
  - Assigning sub-graphs to worker nodes

Stage 2: Optimisation

- Gateway executes a chain of optimisers
- Extension point for developers to plug in their own implementations
  - Handle specific cases
  - Example: Chee Sun's measurement-based optimiser for resource allocation


use dispel.dat..
PE<Validator> ...
= makeCrossVal..
...
Results r = new ..
... => r.input;
submit r;
DISPEL
DISPEL
Language
Processing
"AnnotationDb"
source
expression
SQL Query
data
["class", "uri"]
columns
input
Tuple Burst
struct
outputs
"SELECT euaxssay_id, case ..."
classType
data
Fisher Ratio
output
featureNumber
l- repeat enough of "http://..." -l
l- repeat enough of 24 -l
url
data
Post To HTTP
filename
struct
inputs
Tuple Build
output
parameters
source
SQL Param Query
"GeneratedFeatureDb"
"SELECT url FROM ..."
expression
["class", "uri"]
columns
data
input
Tuple Burst
struct
outputs
filename
l- repeat enough of "http://..." -l
url
Get From HTTP
data
l- repeat enough of 24 -l
l- repeat enough of "select ..." -l
inputs
Tuple Build
output
parameters
source
SQL Param Update
result
"SelectedFeatureDb"
expression
"INSERT INTO selectedFeature (..."
Graph
Optimisation
"AnnotationDb"
source
expression
SQL Query
data
input
Split
outputs
"SELECT euaxssay_id, case ..."
["class", "uri"]
columns
input
Tuple Burst
struct
outputs
struct
inputs
Tuple Build
output
parameters
source
SQL Param Query
"GeneratedFeatureDb"
expression
"SELECT url FROM ..."
["class", "uri"]
columns
data
input
Tuple Burst
struct
outputs
l- repeat enough of "http://..." -l
url
filename
Get From HTTP
classType
data
StdDev Calculate
result
data
...
...
["class", "uri"]
columns
input
Tuple Burst
struct
outputs
struct
inputs
Tuple Build
output
parameters
source
SQL Param Query
"GeneratedFeatureDb"
expression
"SELECT url FROM ..."
["class", "uri"]
columns
data
input
Tuple Burst
struct
outputs
l- repeat enough of "http://..." -l
url
filename
Get From HTTP
classType
data
StdDev Calculate
result
data
data
...
StdDev Combine
result
data
Fisher Ratio Parallel
output
featureNumber
l- repeat enough of 24 -l
l- repeat enough of "http://..." -l
url
data
Post To HTTP
filename
l- repeat enough of 24 -l
l- repeat enough of "select ..." -l
struct
inputs
Tuple Build
output
parameters
source
SQL Param Update
result
"SelectedFeatureDb"
expression
"INSERT INTO selectedFeature (..."
Graph'
Mapping
Concrete
Workflows
Deployment
DIVMp
DIVMq
DIVMr
DIVMs
Execution
Engines

Stage 3: Deployment

- Creating concrete (executable) workflows
  - Mapping PEs to concrete processing units
  - Linking data connections between PEs
  - Configuring data transfers between execution engines (data marshalling)
- Distributed processing
  - Single vs. multiple execution engines
  - Single vs. multiple gateway

Stage 4: Execution and Control

- Processing control
  - Distribute sub-processes to worker nodes
  - Monitor activity of distributed processes
- Data-flow control
  - Initiate and manage data transfer between nodes
  - Publish data sources for result retrieval
- Termination, error management and cleanup
  - Collect and publish progress reports and errors
  - Best effort for cleaning up resources