Interoperability of heterogeneous large-scale scientific workflows and data resources.

Tamas Kukla

School of Electronics and Computer Science

Interoperability of Heterogeneous Large-Scale Scientific Workflows and Data Resources

Tamas Kukla

A thesis submitted in partial fulfilment of the requirements of the University of Westminster for the degree of Doctor of Philosophy

January, 2011

Abstract

Workflow allows e-Scientists to express their experimental processes in a structured way and provides a glue to integrate remote applications. Since Grid provides an enormously large amount of data and computational resources, executing workflows on the Grid results in significant performance improvement. Several workflow management systems, which are widely used by different scientific communities, were developed for various purposes. Therefore, they differ in several aspects.

This thesis outlines two major problems of existing workflow systems: workflow interoperability and data access. On the one hand, existing workflow systems are based on different technologies. Therefore, to achieve interoperability between their workflows at any level is a challenging task. In spite of the fact that there is a clear demand for interoperable workflows, for example, to enable scientists to share workflows, to leverage existing work of others, and to create multi-disciplinary workflows; currently, there are only limited, ad-hoc workflow interoperability solutions available for scientists. Existing solutions only realise workflow interoperability between a small set of workflow systems and do not consider performance issues that arise in the case of large-scale (computational and/or data intensive) scientific workflows. Scientific workflows are typically computation and/or data intensive and are executed in a distributed environment to speed up their execution time. Therefore, their performance is a key issue. Existing interoperability solutions bottleneck the communication between workflows in most scenarios dramatically increasing execu-

tion time. On the other hand, many scientific computational experiments are based on data that reside in data resources which can be of different types and vendors. Many workflow systems support access to limited subsets of such data resources preventing data level workflow interoperation between different systems. Therefore, there is a demand for a general solution that provides access to a wide range of data resources of different types and vendors. If such a solution is general, in the sense that it can be adopted by several workflow systems, then it also enables workflows of different systems to access the same data resources and therefore interoperate at data level. Note that data semantics are out of the scope of this work. For the same reasons as described above, the performance characteristics of such a solution are inevitably important. Although in terms of functionality, there are solutions which could be adopted by workflow systems for this purpose, they provide poor performance. For that reason, they did not gain wide acceptance by the scientific workflow community.

Addressing these issues, a set of architectures is proposed to realise heterogeneous data access and heterogeneous workflow execution solutions. The primary goal was to investigate how such solutions can be implemented and integrated with workflow systems. The secondary aim was to analyse how such solutions can be implemented and utilised by single applications.

Acknowledgements

Even though only a single name is listed on the cover of this thesis, many people contributed to it. I would like to use this section to acknowledge these contributions.

I wish to express a particular gratitude to my research supervisors: Dr. Gabor Z. Terstyanszky, Prof. Stephen C. Winter, and Prof. Peter Kacsuk whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject.

I would like to thank my colleague, Tamas Kiss for his assistance, endless support, suggestions and ideas which contributed to the research especially during the initial steps of my doctoral studies.

I would like to thank the University of Westminster research committee for admitting me to the PhD programme, granting me the research scholarship and allowing me to carry out this research.

Last but not least, I thank my family, friends and colleagues for supporting me through this work and who, in their own particular way, contributed to the success of this long journey.

Tamas Kukla

London, United Kingdom

January 2011

Contents

Abstract ii

Acknowledgements iii

Contents v

List of definitions xii

List of lemmas xix

List of figures xx

List of tables xxiii

Notations xxvii

Abbreviations xxix

1 Introduction **1**

1.1 General overview of Grid 1

1.1.1 Grid Computing 1

1.1.2 Grid infrastructures and middleware 2

1.1.3 Grid Interoperability 2

1.2 Overview of Workflows 3

1.2.1 Workflow definition 4
1.2.2 Workflow specification and structure 5
1.2.3 Workflow execution 5
1.3 Heterogeneity of workflow systems 7
1.4 Workflow interoperability 11
1.4.1 Approaches to workflow interoperability 12
1.5 Interoperability of workflows and data resources 15
1.6 Research overview 16
1.6.1 Research method 18
1.6.2 Contributions 21
1.6.3 Thesis structure 24

2 Heterogeneous Data Access Solutions for Grid applications (DASG) 25

2.1 Key DASG properties and requirements 26
2.2 DASG architecture definition 28
2.2.1 DASG node 28
2.2.2 DASG structure 32
2.2.3 DASG data flow 35
2.2.4 DASG resource 40
2.2.5 DASG interface 42
2.2.6 DASG architecture and solution 43

2.3 DASG architecture analysis 44
2.3.1 DASG generality and extendibility 44
2.3.2 DASG performance 45
2.3.3 DASG bulk data flow 55
2.3.4 DASG DRC flow 58
2.3.5 Recommended DASG structure layout, data flow, and resource layout 60
2.4 Existing and proposed DASG solutions 64
2.4.1 Existing DASG solutions 64
2.4.2 Proposed DASG solutions 67
2.4.3 Comparison of existing and proposed DASG solutions 67
2.5 Implementation 70
2.6 Summary 73

3 Heterogeneous Data Access Solutions for Workflows (DASW) 75
3.1 Key DASW properties and requirements 76
3.2 DASW architecture definition 77
3.2.1 DASW node 78
3.2.2 DASW structure 79
3.2.3 DASW data flow 80
3.2.4 DASW resource 82
3.2.5 DASW interface 82

3.2.6 DASW integration 82
3.2.7 DASW architecture and solution 84
3.3 DASW architecture analysis 84
3.3.1 DASW generality, extendibility, and data access 84
3.3.2 DASW performance 86
3.3.3 DASW bulk data flow 89
3.3.4 DASW DRC flow 91
3.3.5 Recommended DASW structure layout, data flow, and resource layout 93
3.4 Existing and proposed DASW solutions 98
3.4.1 Existing DASW solutions 98
3.4.2 Proposed DASW solutions 104
3.4.3 Comparison of existing and proposed DASW solutions 104
3.5 Implementation 106
3.6 Summary 108

4 Heterogeneous Workflow Execution Solutions for Applications (WESA) 110

4.1 Key WESA properties and requirements 111
4.2 WESA architecture definition 112
4.2.1 WESA node 113
4.2.2 WESA Structure 115

4.2.3 WESA Data flow 116
4.2.4 WESA resources 118
4.2.5 WESA interface 118
4.2.6 WESA architecture and solution 119
4.3 WESA architecture analysis 120
4.3.1 WESA generality and extendibility 120
4.3.2 WESA performance 121
4.3.3 WESA bulk data flow 124
4.3.4 WESA engine and workflow flow 127
4.3.5 Recommended WESA structure layout, data flow, and resource layout 128
4.4 Existing and proposed WESA solutions 136
4.4.1 Existing WESA solution 136
4.4.2 Proposed WESA solutions 139
4.4.3 Comparison of existing and proposed WESA solutions 140
4.5 Implementation 143
4.6 Summary 147

5 Heterogeneous Workflow Execution Solutions for Workflows (WESW) - workflow nesting 150

5.1 Key WESW properties and requirements 151
5.2 WESW architecture definition 151

5.2.1 WESW structure 152
5.2.2 WESW structure 154
5.2.3 WESW data flow 154
5.2.4 WESW resources 156
5.2.5 WESW interface 157
5.2.6 WESW integration 157
5.2.7 WESW architecture and solution 158
5.3 WESW architecture analysis 158
5.3.1 WESW generality, extendibility, and invocation 158
5.3.2 WESW performance 159
5.3.3 WESW bulk data flow 162
5.3.4 WESW engine and workflow flow 168
5.3.5 Recommended WESW structure layout, data flow, and resource layout 171
5.4 Existing and proposed WESW solutions 180
5.4.1 Existing WESW solutions 180
5.4.2 Proposed WESW solutions 185
5.4.3 Comparison of existing and proposed WESW solutions 186
5.5 Implementation 188
5.6 Summary 191

Conclusions **193**

Dissemination of the research findings **200**

A Proofs **202**

Bibliography **212**

List of definitions

Definition 2.1 Node and node type * 28
Definition 2.2 DASG node types 29
Definition 2.3 Instance * 30
Definition 2.4 DASG Instance 30
Definition 2.5 Coexistence of multiple instances * 30
Definition 2.6 Bijection between node types and instances * 31
Definition 2.7 Bijection between DASG node types and instances 31
Definition 2.8 Node type set * 31
Definition 2.9 Coupling * 32
Definition 2.10 Structure * 32
Definition 2.11 Instance layout * 32
Definition 2.12 DASG instance layout 33
Definition 2.13 Type layout * 33
Definition 2.14 DASG type layout 34
Definition 2.15 Structure layout * 34

Definition 2.16 DASG structure layout 34

Definition 2.17 DASG data types 35

Definition 2.18 Path * 36

Definition 2.19 Path layout * 36

Definition 2.20 DASG DRC path layout 37

Definition 2.21 DASG bulk data path types 37

Definition 2.22 Mapping between path layouts and paths * 37

Definition 2.23 Byte array and its length * 38

Definition 2.24 Byte array concatenation * 38

Definition 2.25 Transferring a byte array via a path * 38

Definition 2.26 Transferring a sequence of byte arrays via a path * 38

Definition 2.27 Pipelined transfer * 39

Definition 2.28 Non pipelined transfer * 39

Definition 2.29 DASG DRC staging 39

Definition 2.30 DASG bulk data staging 39

Definition 2.31 Set of DASG data flow types 39

Definition 2.32 Resource types * 40

Definition 2.33 Resource layout * 41

Definition 2.34 DASG resource layout 41

Definition 2.35 Interface representation * 42

Definition 2.36 Interface generality * 42

Definition 2.37 DASG frontend interface 42

Definition 2.38 DASG backend interface 43

Definition 2.39 Set of possible DASG interfaces 43

Definition 2.40 Set of possible DASG architectures 43

Definition 2.41 DASG solution 44

Definition 2.42 DASG scenarios 45

Definition 2.43 DASG scenario execution 47

Definition 2.44 Port functions * 48

Definition 2.45 Time of byte array transfer between two nodes * 49

Definition 2.46 Time and latency of pipelined transfer * 50

Definition 2.47 Time and latency of non pipelined transfer * 51

Definition 2.48 Simultaneous transfer via a path layout * 52

Definition 2.49 Performance characteristics of simultaneous transfer * 53

Definition 2.50 Performance function of DASG DRC transfer 54

Definition 2.51 Performance functions of DASG bulk data transfer 54

Definition 2.52 Overall DASG performance functions 54

Definition 2.53 Scalability of DASG data transfer 55

Definition 2.54 Data flow case * 55

Definition 2.55 Data flow case equivalence * 56

Definition 3.1 DASW nodes and node types 78

Definition 3.2 DASW instance 78

Definition 3.3 Bijection between DASW node types and instances 78
Definition 3.4 DASW instance layout 79
Definition 3.5 DASW type layout 80
Definition 3.6 DASW structure layout 80
Definition 3.7 DASW Data types 80
Definition 3.8 DASW DRC flow 81
Definition 3.9 DASW bulk data flow 81
Definition 3.10 DASW data flow types 81
Definition 3.11 DASW resource layout 82
Definition 3.12 DASW Interfaces 82
Definition 3.13 DASW Subject of integration 82
Definition 3.14 Request representation 83
Definition 3.15 Set of possible DASW integrations 83
Definition 3.16 Set of DASW architectures 84
Definition 3.17 DASW solution 84
Definition 3.18 DASW scenarios 86
Definition 3.19 DASW scenario execution 87
Definition 3.20 Performance function of DASW DRC transfer 87
Definition 3.21 Performance functions of DASW bulk data transfer 88
Definition 3.22 Overall DASW performance functions 88
Definition 3.23 Scalability of DASW data transfer 88

Definition 4.1 WESA nodes and node types . . . 113
Definition 4.2 WESA Instance . . . 114
Definition 4.3 Bijection between WESA node types and instances . . . 114
Definition 4.4 WESA instance layout . . . 115
Definition 4.5 WESA type layout . . . 115
Definition 4.6 WESA structure layout . . . 115
Definition 4.7 WESA data types . . . 116
Definition 4.8 WESA engine data flow . . . 116
Definition 4.9 WESA workflow data flow . . . 117
Definition 4.10 WESA bulk data flow . . . 117
Definition 4.11 WESA data flow types . . . 117
Definition 4.12 WESA resource layout . . . 118
Definition 4.13 WESA frontend interface . . . 118
Definition 4.14 WESA backend interface . . . 119
Definition 4.15 Set of possible WESA interfaces . . . 119
Definition 4.16 Set of possible WESA architectures . . . 119
Definition 4.17 WESA solution . . . 120
Definition 4.18 WESA scenarios . . . 121
Definition 4.19 WESA scenario execution . . . 122
Definition 4.20 Performance of WESA engine and workflow transfer . . . 123
Definition 4.21 Performance of WESA bulk data transfer . . . 123

Definition 4.22 Overall WESA performance functions 124

Definition 4.23 Scalability of WESA data transfer 124

Definition 5.1 WESW nodes and node types 152

Definition 5.2 WESW Instance 153

Definition 5.3 Bijection between WESW node types and instances 153

Definition 5.4 WESW instance layout 154

Definition 5.5 WESW type layout 154

Definition 5.6 WESW structure layout 154

Definition 5.7 WESW data types 154

Definition 5.8 WESW engine data flow 155

Definition 5.9 WESW workflow data flow 155

Definition 5.10 WESW bulk data flow 156

Definition 5.11 WESW data flow types 156

Definition 5.12 WESW resource layout 156

Definition 5.13 WESW interfaces 157

Definition 5.14 WESW Subject of integration 157

Definition 5.15 Set of possible WESW architectures 158

Definition 5.16 WESW solution 158

Definition 5.17 WESW scenarios 159

Definition 5.18 WESW scenario execution 160

Definition 5.19 Performance of WESW engine and workflow transfer 161

Definition 5.20 Performance of WESW bulk data transfer 161

Definition 5.21 Overall WESW performance functions 161

Definition 5.22 Scalability of WESW data transfer 162

Definition A.1 Structure generated by an instance layout 202

Definition A.2 Structure generated by a type layout 202

List of lemmas

Lemma 2.1 Time of transferring a byte array sequence via a path * 50

Lemma 2.2 Slice size independence * 51

Lemma 2.3 Performance characteristics of simultaneous transfer * 53

Lemma 2.4 Simultaneous transfer of equivalent data flow cases * 56

Lemma A.1 Structure layout implementation 203

Lemma A.2 Time of transferring a byte array sequence via a path * 205

Lemma A.3 Slice size independence 207

Lemma A.4 Performance characteristics of simultaneous transfer 208

Lemma A.5 Simultaneous transfer of equivalent data flow cases 209

List of Figures

1.1 Growth in the content of the MyExperiment workflow repository . . . 4

1.2 Different workflow abstraction levels 6

1.3 Heterogeneous technologies in current workflow systems 9

1.4 Research process 19

1.5 Architecture analysis and evaluation 20

1.6 Contributions and their relations 22

2.1 Concept of existing DASGs 25

2.2 Nodes of existing DASGs 29

2.3 Example DASG structure 35

2.4 Examples for each data transfer case 50

2.5 Structure and dataflow of existing and proposed DASG architectures 65

2.6 DASG overhead predictions 69

2.7 Deploying MySQL client using the GEMLCA Administration Portlet 72

2.8 DASG implementation based on GEMLCA 72

3.1 Concept of existing DASWs 75
3.2 DASW Data access approaches 77
3.3 DASW node types 79
3.4 Request representation within workflows 83
3.5 Subject of integration 85
3.6 Combinations of recommended DASW data flow cases 96
3.7 Structure and data flow of existing DASW architectures 100
3.8 Parametrisation of a MySQL client in a P-GRADE workflow 106
3.9 DASW implementation based on GEMLCA 107

4.1 WESA concept 111
4.2 WESA node types 114
4.3 Combinations of recommended WESA data flow cases 132
4.4 Existing and proposed WESA architectures 136
4.5 WESA overhead predictions 141
4.6 Deploying Taverna using the GEMLCA Admin. Portlet 144
4.7 Implementation of WESA PC10 based on GEMLCA 145
4.8 Implementation of WESA PC3 based on GEMLCA 146
4.9 Implementation of WESA PC16 based on GEMLCA 146

5.1 WESW concept 150
5.2 WESW Workflow invocation types 152

5.3 WESW node types 153

5.4 Combinations of recommended WESW data flow cases 174

5.5 Existing and proposed WESW architectures 182

5.6 Parametrisation of a Triana child workflow in a P-GRADE parent workflow 189

5.7 Implementation of WESW PC26 based on GEMLCA 190

5.8 Implementation of WESW PC5 based on GEMLCA 190

5.9 Implementation of WESW PC31 based on GEMLCA 191

List of Tables

1.1 Data resource support in current workflow systems 10

2.1 DASG node matrix 31

2.2 Properties of different resource types 40

2.3 DASG bulk data flow cases 57

2.4 Time of DASG bulk data transfer 57

2.5 Overhead and scalability of DASG bulk data staging 57

2.6 Latency and scalability of DASG bulk data staging 58

2.7 DASG DRC data flow cases 59

2.8 Time and scalability of DASG DRC transfer 59

2.9 Elimination of DASG data flow cases 62

2.10 Recommended DASG structure, data flow, and resource layout 63

2.11 Analysis of proposed and existing DASG architectures 65

3.1 DASW node matrix 79

3.2 DASW of bulk data flow cases 89

3.3 Overhead and scalability of DASW bulk data staging 90

3.4 Latency and scalability of DASW bulk data staging 90

3.5 Time of DASW bulk data transfer 91

3.6 DASW DRC data flow cases 91

3.7 Transfer time and scalability of DASW DRC transfer 92

3.8 Elimination of DASW data flow cases 94

3.9 Proposed DASW structures, data flows, and resource layouts 95

3.10 Proposed DASW structure and data flow combinations in different cases 98

3.11 Performance of proposed DASW structures and data flows 99

3.12 Existing and proposed DASW architectures 101

3.13 Performance of the existing DASW architectures part 1/2. 102

3.14 Performance of the existing DASW architectures part 2/2. 102

4.1 WESA node matrix 115

4.2 Size of different workflow engines 117

4.3 WESA bulk data flow cases 125

4.4 Time of WESA bulk data transfer 125

4.5 Overhead and scalability of WESA bulk data staging 126

4.6 Latency and scalability of WESA bulk data staging 126

4.7 WESA engine data flow cases 127

4.8 Transfer time and scalability of WESA engine transfer 127

4.9 WESA workflow data flow cases 128

4.10 Transfer time and scalability of WESA workflow transfer 128

4.11 Elimination of WESA data flow cases 130

4.12 Proposed WESA structures, data flows, and resource layouts 131

4.13 Proposed WESA structure and data flow combinations in different cases 134

4.14 Performance of proposed WESA structures and data flows 135

4.15 Existing and proposed WESA architectures 137

4.16 Performance of the Gria Service based architectures part 1/2. 138

4.17 Performance of the Gria Service based architectures part 2/2. 138

5.1 WESW node matrix 154

5.2 Time of WESW bulk data transfer 163

5.3 Overhead and scalability of WESW bulk data staging 163

5.4 Latency and scalability of WESW bulk data staging 164

5.5 WESW bulk data flow cases part 1/3. 165

5.6 WESW bulk data flow cases part 2/3. 166

5.7 WESW bulk data flow cases part 3/3. 167

5.8 WESW engine data flow cases 168

5.9 WESW workflow data flow cases 169

5.10 Time and scalability of WESW engine transfer 170

5.11 Time and scalability of WESW workflow transfer 170

5.12 Elimination of WESW data flow cases 172

5.13 Proposed WESW structures, data flows, and resource layouts 173

5.14 Proposed WESW structure and data flow combinations in different cases 178

5.15 Performance characteristics of proposed WESW structure and data data flow combinations part 1/2. 179

5.16 Performance characteristics of proposed WESW structure and data data flow combinations part 2/2. 180

5.17 Existing and proposed WESW architectures 181

5.18 Performance characteristics of the Gria and WFBus-VRE Service based architectures part 1/4. 183

5.19 Performance characteristics of the Gria and WFBus-VRE Service based architectures part 2/4. 184

5.20 Performance characteristics of the Gria and WFBus-VRE Service based architectures part 3/4. 184

5.21 Performance characteristics of the Gria and WFBus-VRE Service based architectures part 4/4. 185

5.22 Performance comparison of the proposed and existing WESW architectures 187

5.23 Number of proposed and implemented architecture sets 196

Notations

$\mathbb{N}$	set of natural numbers including 0
$\mathbb{N}^+$	set of positive natural numbers
$\mathbb{R}$	set of real numbers
$\mathbb{R}^+$	set of positive real numbers
$\mathbb{R}_0^+$	set of non-negative real numbers
$\mathbb{L}$	set of logical values: 0 and 1
$\mathbb{B}$	set of bytes, where $\mathbb{B} = \mathbb{L}^8$
$\forall$	for all
$\exists$	exists
$\exists!$	uniquely exists
$\in$	element
$\{x \in X \parallel \theta(x)\}$	a subset of X whose members are satisfying formula $\theta(x)$
$(x_1, x_2, \ldots, x_n)$	ordered n-tuple
$X_1 \times X_2 \times \cdots \times X_n$	Cartesian product of sets $X_1, X_2, \ldots, X_n$
$\lvert X \rvert$	cardinality of a set X
$\wedge$	logical and operation

$\vee$	logical or operation
$\neg$	logical negation operation
$\Rightarrow$	logical implication operation $(x \Rightarrow y) = (\neg x \vee y)$
$\Leftrightarrow$	logical equivalence operation
$[n..m]$	$\{x \in \mathbb{N} \| n \geqq x \geqq m\}(n, m) \in \mathbb{N}$
$\overline{b}$	transitive, symmetric, reflexive closure of binary relation b
$\subset$	subset
$\mathscr{P}$	power set
$\varnothing$	empty set
$\lceil r \rceil$	ceiling function: smallest integer not smaller than $r \in \mathbb{R}$
$\lfloor r \rfloor$	floor function: largest integer not greater than $r \in \mathbb{R}$
$\mathrm{dom}(h)$	domain of function h
$\llbracket h \rrbracket$	truth set of logical function h conatining all elements of $\mathrm{dom}(h)$ where h is true
$\circ$	composition of binary relations
$[X \to Y]$	set of all functions from a set X to a set Y
$\chi(\theta)$	function that maps 1 to logical statement θ if it is true and maps 0 otherwise

Abbreviations

DASG	heterogeneous Data Access Solution for Grid applications
DASW	heterogeneous Data Access Solution for Workflows
WESA	heterogeneous Workflow Execution Solution for Applications
WESW	heterogeneous Workflow Execution Solution for Workflows
DRC	Data Resource Client
DAG	Directed Acyclic Graph
DCG	Directed Cyclic Graph
API	Application Programming Interface
CLI	Command Line Interface

Chapter 1

Introduction

1.1 General overview of Grid

1.1.1 Grid Computing

"Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation" [1]. The *Grid* is an infrastructure of computers, databases, networks and scientific instruments that belong to multiple organisations. Since applications on the Grid often use large amounts of data and secure access to different kinds of resources, managing these applications is a complex task. For this reason, different high-level tools have been developed to ease the usage of Grid resources and help the composition and orchestration of low-level tasks.

1.1.2 Grid infrastructures and middleware

Many different Grid infrastructures, with large user communities, emerged during the last decade. TeraGrid [2] of the American National Science Foundation (NSF) interconnects the institutes of the American National Centre for Supercomputing Applications forming one of the largest Grid-based infrastructure (about 30000 nodes [3]). The UK National Grid Service [4] (NGS) provides also access to numerous computer resources (about 1400 CPUs [5]) hosted by the University of Bristol, Cardiff, Leeds, Leicester, Manchester, Oxford, Westminster, etc. Enabling Grids for E-sciencE [6] (EGEE) is the largest Grid infrastructure. More than 100 institutions are part of it from more than 50 countries all over the world providing about 110,000 CPUs. [7] Distributed European Infrastructure for Supercomputing Applications [8] (DEISA) is a European wide Grid infrastructure hosting about 30,000 processors [9] and connecting 11 European national supercomputing centres.

These Grid infrastructures are based on different *Grid middleware*, which is the software layer between computer resources and users. TeraGrid and NGS are based on the Globus Toolkit [10, 11], EGEE is based on gLite [12] and DEISA is based on Unicore [13, 14] and the Globus Toolkit.

1.1.3 Grid Interoperability

Different Grid middleware provides different types of computational and data resources. (See comparison of different Grid middleware technologies [15, 16].) For this reason, there are non-interoperable Grid islands which inhibits the inter-organisational usage of different Grid resources [17]. *Grid interoperability* is the ability of different Grids to cooperate and mutually share their resources. Grid communities put a lot of effort nowadays to attain interoperability. (See Workflow Level Interoperation of Grid Data Resources [18], that identifies different interoperation levels

of Grid resources and summarises the current state of the art.)

The Open Grid Forum's (OGF) Grid Interoperation Now Community Group (GIN) coordinates a significant project that is aiming to achieve interoperability between different Grid systems. Their goal is to implement interoperation on specific topics such as information services, data movement, job submission and authorisation and identity. This research group distinguishes interoperability and interoperation.

The Simple API for Grid Application (SAGA) project [19] focuses on Grid interoperability at application level by providing a standardized interface that comprises method calls for performing the most commonly needed Grid-functionality. Both SAGA and GIN use the same working-definitions and distinguish between interoperability and interoperation.

According to the definition in [20] both *interoperability* and *interoperation* are trying to make different infrastructures work together, but while the former is a long-term solution based on standards, which have to be adopted by the infrastructures, the latter is rather a short-term solution and does not require changes in the infrastructures. These two notions are not distinguished in this document as they are not distinguished in most of the publications which this document cites.

1.2 Overview of Workflows

Scientific workflows are widely used by the Grid community to automate large-scale, computationally intensive experiments. Communities of various research areas, such as bioinformatics, physics, geographics, astronomy, proposed and developed different workflow management systems in the last decade.

According to the "Case Studies in Workflow Fragment Reuse" in [22], few hun-

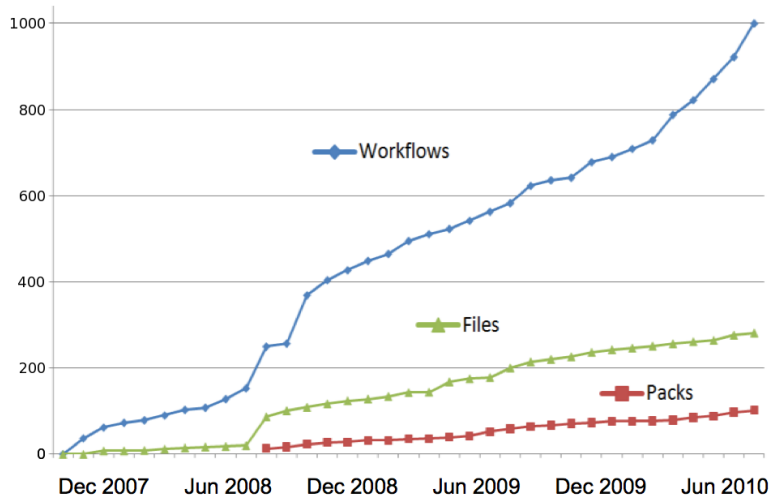

Figure 1.1: Growth in the content of the MyExperiment workflow repository [21].

dred users of different workflow systems created several hundred workflows up to 2005. Figure 1.1 illustrates the number of workflows available in a single workflow repository called MyExperiment over the past few years. As the figure suggests, the number of publicly available workflows is growing continuously.

1.2.1 Workflow definition

A general workflow definition based on the Workflow Management Coalition's "The Workflow Reference Model" [23] is the following: "*Workflow* is concerned with the automation of procedures whereby files and data are passed between participants according to a defined set of rules to achieve an overall goal." Similar definition of scientific workflow can be found in [24]. A definition of Grid workflow can be found in [25], which states that "A *Grid workflow* is a workflow within a Grid computing environment." Executing workflows on the Grid results significant performance improvement and offers the ability to execute multiple tasks concurrently in a distributed environment. However, because of the diversity of different Grid resources

and complexity of Grid infrastructures, Grid workflows are typically less reliable and complicated to execute.

1.2.2 Workflow specification and structure

A *workflow specification* can be abstract or concrete. The former describes the workflow structure, but does not map the tasks to concrete Grid resources, while the latter is rather an executable form, so it includes all the data which is required for execution, including resource mappings and workflow inputs as well. There are systems in which workflows are defined and stored in abstract form and the concrete form is generated before or during workflow execution, while other systems simply store the executable concrete form. However, further abstraction levels can be identified. Figure 1.2, illustrates these many abstraction levels supported by a single workflow language called GWorkflowDL [26, 27].

According to taxonomies given in [28] and [24], a workflow structure can be represented as a DAG[1] or non-DAG. The difference is that non-DAG workflows allow iterations of a subset of tasks (this structure is usually called DCG[2]), workflow recursion, and/or further control constraints.

1.2.3 Workflow execution

The *workflow scheduler* is responsible for controlling the workflow execution process taking into account that the tasks (jobs) have to be executed in a given order. The control can be either: centralised, where one scheduler controls the whole execution process; hierarchical, where the central scheduler gives subtasks (sub-workflows) to

[1]Directed Acyclic Graph
[2]Directed Cyclic Graph

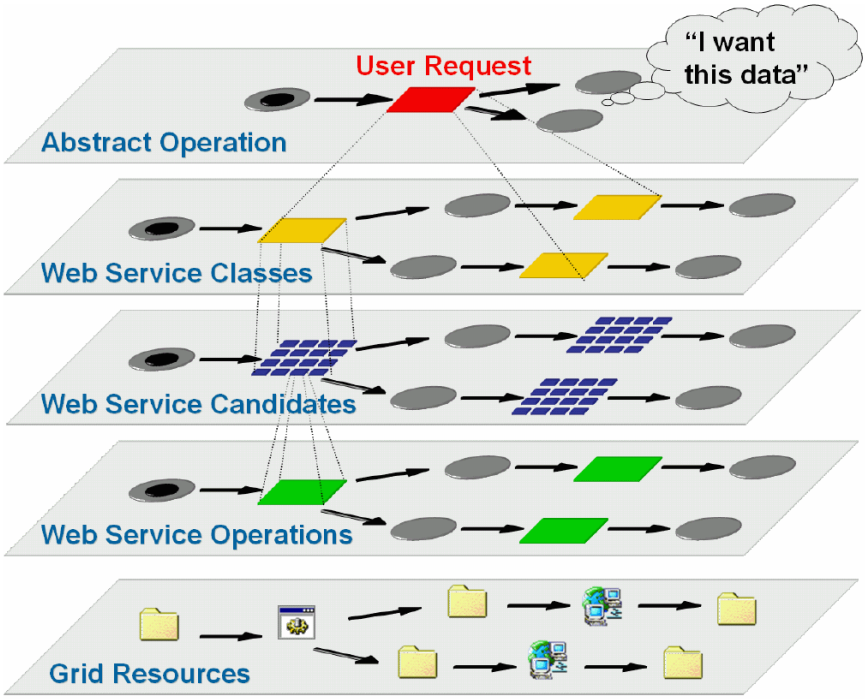

Figure 1.2: Different workflow abstraction levels supported by GWorkflowDL workflow language [26]

lower-level schedulers; or decentralised, where multiple schedulers are conducting the sub-workflow execution without being controlled by a central scheduler.

Before workflow task execution, the executable jobs and/or their requisite data are transferred to different Grid nodes. However, in some cases the executable jobs are already deployed on Grid nodes. In this case no task transfer is needed. After successful execution, the result is gathered and can be used as an input of the following tasks. This means that intermediate data has to be moved during execution from one Grid node to another. According to the taxonomy given in [28] and [24], data movement can be either: centralised, where a central mechanism moves the data from one node to the other; mediated, where data transfer is managed by a distributed system; or peer-to-peer, where data is directly passed between the computational resources. Furthermore, workflow management systems provide fault

handling mechanisms, in order to handle workflow execution failures. This also can be handled at different levels such as at task or at workflow level.

The system which is responsible for the whole workflow execution process including workflow scheduling, data movement, fault tolerance and monitoring is called the *workflow engine*. This engine is part of the *workflow management system* that typically provides a user interface for workflow development, execution, fault handling, monitoring and other workflow related functionalities. In the case of Grid workflow management system, this is connected to at least one Grid, which provides the computational resources for task execution.

1.3 Heterogeneity of workflow systems

Since workflow management systems were developed for various purposes, they differ in several aspects and they are based on different technologies as described in [29], [28] and [24]. The most important differences are summarised in this section and illustrated in figure 1.3[3] and table 1.1.

Triana [30, 31, 32] is a lightweight, modular, general purpose workflow based distributed problem solving environment developed at Cardiff University. It has been used in numerous projects such as GridOneD [33], GridLab [34]. *Taverna* [35, 36, 37] was mainly developed as a collaboration of the University of Manchester and Southampton so as to create a high-level tool for bioinformatics workflow orchestration and execution. The project is, among others, founded by the Open Middleware Infrastructure Institute UK (OMII-UK), Engineering and Physical Sciences Research Council (EPSRC) and Microsoft. The *P-GRADE portal* [38, 39] is a web-based high-level tool for workflow development, execution and monitoring. The

[3]This figure has been used by the SHIWA project in various documents and presentations.

portal, which was developed by the MTA SZTAKI Research Institute and Centre for Parallel Computing at University of Westminster, is the official portal of several European Grids, such as EGEE, VOCE, See-Grid or Hun-Grid and provides access to numerous types of different Grid data resources, for example: OGSA-DAI or SRB. *Askalon* [40], which was developed at University of Innsbruck, is a programming environment and tool-set for cluster and Grid programming. It provides UML workflow modelling tool, XML based abstract workflow language and workflow scheduling optimisation methods. *Kepler* [41], which has been developed as a cross-project collaboration, provides a Grid-based workflow system for scientists from different areas, such as astronomy, biology or ecology. It is based on Ptolemy II [42], which has been developed at University of California - Berkeley. The development of the *UNICORE* (UNiform Interface to COmputing Resources) middleware started as two German projects funded by the German ministry of education and research. The UNICORE workflow system was developed based on this middleware. *Pegasus* [43] (Planning for Execution on Grids) is a framework for mapping computational workflows on distributed resources. It enables users to represent workflows at an abstract level while hides the target execution resources. *K-Wf Grid* [26] (Knowledge-based Workflow System for Grid Applications) adopts the approaches of semantic Web to create a knowledge-based system, that is able to utilise the dynamically changing, complex Grid environments. *MOTEUR* [44] workflow engine is aiming to realise a system that allows both a simple description of the data flow and efficient execution on the Grid.

Most systems are coupled with one workflow engine. Taverna uses Freefluo, Triana uses Triana Engine, K-Wf Grid [26] uses GWES [26] (Grid Workflow Execution Service). UNICORE uses the Shark [45] open source workflow engine. Older versions of P-GRADE used Condor DAGMan [46], while its recent version uses its own engine called Xen.

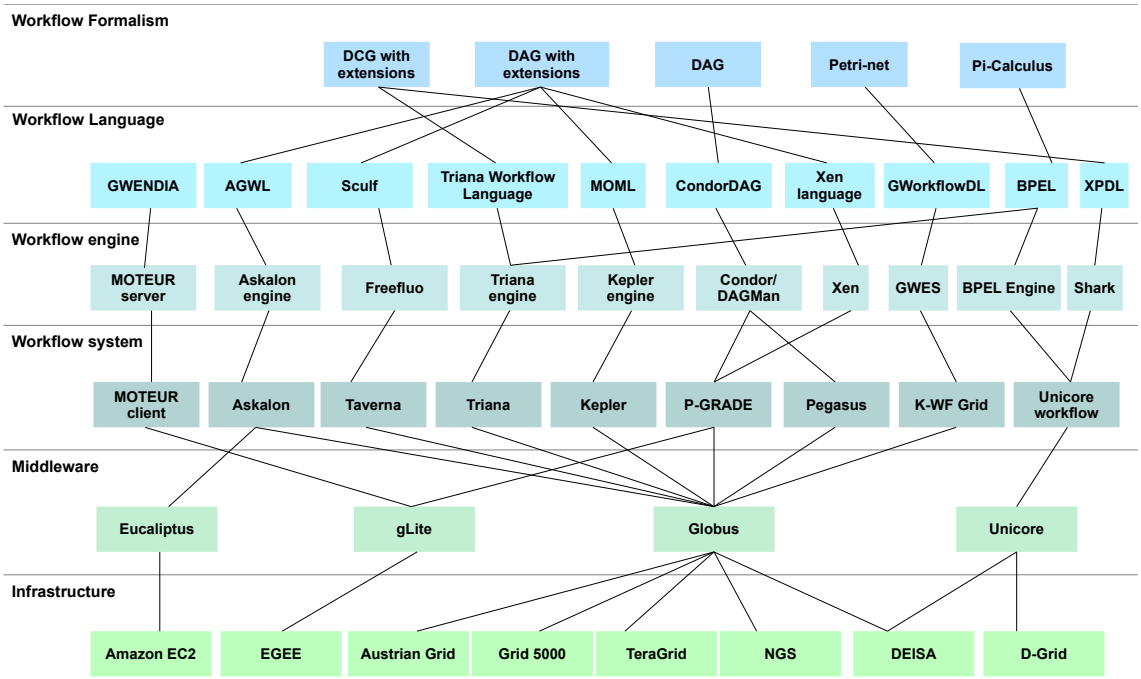

Figure 1.3: Heterogeneous technologies in current workflow systems

Many workflow systems use dissimilar workflow description languages. While Triana is able to interpret BPEL [47] (Business Process Execution Language), its own defined language and additional workflow formats (since its workflow interpreter is extendible), most systems are restricted to one language. Taverna workflows are represented in Scufl [48] (Simple Conceptual Unified Flow Language), older versions of P-GRADE used Condor DAG, now it uses its own defined format (Xen language), Kepler uses MOML [49] (Modeling Markup Language), while K-WfGrid uses GWorkflowDL [27] (Grid Workflow Description Language). Because of the diversity of workflow languages, scientists who use different workflow systems cannot exchange workflows.

Workflow description languages can be based on various workflow formalisms. Some workflow languages, such as the Condor DAG, use simple directed acyclic graph (DAG) workflow structure that does not allow the usage of loop, recursion or

nested workflow. However Scufl, that is also a DAG based language, is extended with control constraints supporting the usage of if/else, case and loop structures within Taverna workflows. The new version of the P-GRADE portal also uses a DAG based language, which is extended with recursion and workflow nesting. GWorkflowDL is based on Petri Nets [50], while BPEL is Pi-Calculus [51] based. Both Petri Nets and Pi-Calculus have a wider range of expression capabilities, for instance they allow the concept of non-determinism. Because of these differences, it is not a trivial issue to express a workflow of one type in the description language of another. For instance, a Petri Net based workflow cannot always be converted into a DAG based language, since DAG cannot express iteration or non-deterministic choice.

Since most workflow management systems are restricted to use one (Grid) middleware, it might be a problem to reuse jobs of a workflow that was created in another workflow management system, because the executable of the job might not run on other middleware.

		Data resources						
		GridFTP	SRB	LFC	Amazon S3	HTTP	JDBC	OGSA-DAI
Workflow systems	Askalon	x						
	K-WF Grid	x						
	MOTEUR			x		x		
	P-GRADE	x	x	x			x	x
	Pegasus	x	x	x	x	x		
	Triana	x				x	x	

Table 1.1: Data resource support in current workflow systems

An important part of a workflow is the data that it processes and generates. This data can reside in various types of data resources. Table 1.1[4] illustrates the heterogeneity of data resources supported by the different workflow systems. Askalon, K-WF Grid, P-GRADE, Pegasus, and Triana all support GridFTP [52] protocol for

[4]The table has been created based on the information provided directly by the developer teams of the included workflow systems.

transferring files. SRB [53] (Storage Resource Broker) is supported by P-GRADE, LFC [54] (LCG File Catalogue) is supported by P-GRADE, MOTEUR, and Pegasus. The latter two also support gathering data from a given HTTP location and PEGASUS also supports Amazon S3 [55] (Simple Storage Service). Triana provides access to different kinds of databases via JDBC [56] (Java Database Connectivity), while P-GRADE also supports OGSA-DAI [57, 58] (Open Grid Services Architecture Data Access and Integration). However, most workflow systems, such as Askalon, K-WF Grid, MOTEUR, or PEGASUS, do not support access to structured data resources. Since most workflow systems support only a limited set of data resources as illustrated in table 1.1, workflows of different systems cannot access and process data of other workflows if they do not support the same data resource type.

1.4 Workflow interoperability

The Workflow Management Coalition defines *workflow interoperability* in general in [59] as:"The ability for two or more Workflow Engines to communicate and work together to coordinate work." In this definition the workflow engine is a service which provides the workflow run-time environment. Interoperability between workflows can be realised at various levels.

Making workflow systems interoperable and reusing workflows that were developed in different workflow management systems are a natural desire of e-Scientists, because these: enable inter-organizational collaboration between different scientific groups; speed up the design phase; and improve the quality by allowing the usage of already validated workflows even if they were developed within different systems. The Workflow Management Research Group [60] of the OGF (Open Grid Forum) is focusing on workflow sharing and interoperability. The Workflow Management Coalition [61] tries to decrease the risks of using business process management and

workflow products via interoperability standards. The CppWfMS [62] project is aiming to achieve workflow language interoperability by defining interfaces for workflow description translation. The SHIWA project [63] is aiming to achieve workflow interoperability applying both coarse- and fine-grained strategies. Coarse-grained approach treats workflow engines as black-box systems where complete workflows are sent for enactment. The fine-grained approach tries to achieve workflow language interoperability by defining an intermediate workflow representation that can be used for translation across different workflow systems.

1.4.1 Approaches to workflow interoperability

Since workflow systems are based on several technologies, workflow interoperability can be achieved at different levels. Three of these: language level, message level, and engine level interoperability are described in the following sections.

Interoperability at the level of workflow languages

Workflow description language *standardization* would enable users of different workflow management systems to exchange workflows. This would realise interoperability by defining a common workflow description language that has to be adopted by every existing workflow management system. Such a top-down approach requires large efforts and is against a user-centric design, since it does not support scientists in formulating their research activities in their preferred workflow description language. If such a standard format will be defined and accepted, workflow management systems will either adopt this format or define import/export processes for *workflow translation* [64].

Workflow description language *translation* would enable users to reuse workflows

created in different workflow systems using their preferred and familiar environment. Workflow translation can be realised by using an intermediate workflow language representation. YAWL [65] (Yet another workflow language) is based on an extensive analysis of (more than 30) existing workflow systems using a set of workflow patterns described in [66]. Because of its expressive power and formal semantics, YAWL might be a candidate to be used as an intermediate language for workflow translations. See, for instance, BPEL to YAWL translation described in [67]. The CppWfMS workflow system [68], that was developed by CNAF department of the National Institute of Nuclear Physics in Italy, defines interfaces for workflow description translation to achieve workflow language interoperability. It contains a JDL (Job Description Language, that is able to describe simple jobs as well as DAG based workflows) to GWorkflowDL converter and also a Scufl to GWorkflowDL converter, that transforms simple Scufl workflows using XSLT (Extensible Stylesheet Language Transformations).

Both language translation and language standardization would help in enabling the same workflow to be run on different infrastructures. However, having a standard workflow language or workflow translators is not sufficient to achieve interoperability, since jobs, job descriptions, data, and execution environments also have to be standardised or mapped. Furthermore, because of different expression capabilities of workflow languages, it is not always possible to translate one language to another.

Interoperability at the level of message passing

Message level workflow interoperability is the ability of workflows of different systems to exchange information by sending data messages. The Scientific Workflow Interoperability Framework [69, 70] (SWIF) realises workflow interoperability at this level, based on a Publish/Subscribe asynchronous messaging system. Using a set of Web Services that follow WS-Eventing Specifications [71], SWIF makes processes

on a workflow system available to other workflows. The PS-SWIF GUI provides tools for publishing workflow activities and subscribing for those. When a published workflow activity is executed, all subscribed workflows receive a notification message.

This solution is general, builds on existing standards, and can be adopted by any workflow system that supports the invocation of Web Services. However, in order to achieve interoperability between different workflows based on this approach, users should have their own workflow systems installed on their machines and they have to modify these workflows to enable their communication via Web Services.

Interoperability at the level of workflow engines

An alternative approach to attain workflow interoperability is to enable workflow management systems to execute non-native workflows by invoking external workflow engines. The aim of this concept is to enable scientists to create such heterogeneous nested workflows where child workflows of a given workflow system can be embedded into a parent workflow of another system enabling interoperability not only between the parent and child workflow but also between multiple child workflows of different kinds embedded in the parent workflow. Non native child workflows are black boxes to the users of the parent workflow system, they don't have to understand how such a workflow works, and they do not have to modify them.

The SIMDAT [72] project identified an approach to this interoperability level in [73], where the functionality of the different workflow engines is wrapped and published as Web/Grid services. The client passes the workflow written in the appropriate workflow language and the input data to the workflow engine that will execute it and give back the results to the client. This client can be used in any workflow management system for workflow execution. The VLE-WFBus [74] system,

developed by the Dutch Virtual Laboratory for e-Science project, provides a meta workflow system, that encapsulates a few popular workflow engines and allows the composition of high-level heterogeneous workflows via a Vergil based GUI provided by the Ptolemy project [42]. These solutions are detailed in section 5.4.

Although there is a high demand for a solution that enables scientists to connect heterogeneous, interoperable workflows, due to their limitations and ad-hoc implementation (see analysis in section 5.4) none of the above solutions is widely utilised.

1.5 Interoperability of workflows and data resources

Workflow data can be stored in various types of data resources (such as: relational databases, XML databases, file system based resources, data repositories). There are several products in the case of each kind of data resource. For instance: MySQL [75], Oracle database [76], PostgreSQL [77], IBM DB2 [78], and Microsoft SQL Server [79] are relational databases; Xindice [80], eXist [81] are semi-structured databases; SRM [82], SRB [53], FTP [83], and GridFTP [52] provide file system based data resources; D-SPACE repository [84] and Fedora repository [85] are digital repositories for storing data and related metadata. Although many scientific experiments rely on data stored in various data resources, most workflow systems support only a small subset of these and many of them do not provide access to databases at all. See table 1.1. For this reason, scientists have to use different tools before workflow submission to access their datasets and gather the required data on which they want to carry out computational experiments. A general solution for accessing heterogeneous data that can be easily integrated with workflow systems

would help scientists to automate their workflows independently of what resource its data resides.

There are existing solutions that can be used for accessing databases. JDBC defined by Sun Microsystems and ODBC [86] (Open Database Connectivity) defined by the SQL Access Group provide connection to several relational databases to allow the execution of SQL queries using the same interface.

OGSA-DAI, that provides access to a larger set of data resources, can also be used for this purpose [87]. It integrates numerous SQL as well as XML databases, and also file systems. It allows the execution of complex workflows of data related requests, and it is based on Grid infrastructure. It has been utilised by numerous projects in the UK e-Science community and world-wide [88]. As a reference implementation, OGSA-DAI implements WS-DAI, WS-DAIR, and WS-DAIX standard recommendations [89] of the DAIS (Database Access and Integration Services) Working Group, which is part of the OGF (Open Grid Forum).

JDBC and ODBC are widely utilised by the industry, but they are only suitable for accessing relational databases. Although OGSA-DAI is a good candidate to use as a general solution for accessing heterogeneous data resources, it has limitations in terms of performance, especially in the case of large number of requests and large amounts of data. These solutions and further data access solutions are detailed section 2.4 and 3.4.

1.6 Research overview

Research statement Coexistence of different Grid middleware services, data resources and workflow systems encumbers the collaboration of scientific communities especially in the case of multidisciplinary research. In addition, new solutions arise

on a regular basis. Often not only different solutions are not able to interoperate, but different versions of the same software are not compatible either. Because of the heterogeneity of these systems and the dynamic changes of this field, attaining interoperability is a demanding task. To address this issue, flexible and easily extendible interoperability solutions are needed which do not bottleneck the performance of the connected software components.

This thesis focuses on two major problems of currently existing workflow systems: workflow interoperability at the level of workflow engines and data access. It proposes a set of architectures to realise heterogeneous data access solutions and to realise heterogeneous workflow execution solutions. The primary goal was to investigate how such solutions can be implemented and integrated with workflow systems. The secondary aim was to analyse how such solutions can be implemented and utilised by single applications.

Hypothesis By dynamically utilizing Grid resources to support heterogeneous data access and distributed heterogeneous workflow execution solutions, flexible and easily extendible architectures can be constructed which also provide high performance data exchange between different software components. Such architectures can be identified based on a mathematical model, that enables the analysis of a large number of architectures taking under consideration key properties such as generality, extendibility and performance.

This mathematical model is designed to analyse existing solutions of the field and numerous possibilities how they can be improved. The model provides concepts for defining different architectures and provides a set of functions to analyse the performance characteristics of these. The key novelty in the case of most proposed architectures is that computational and storage machines provided by the Grid are utilised in order to divide the load of the different software components. This can be

achieved by dynamically distributing these software components between the available machines. This way highly scalable architectures can be realised that deliver data between these components with low overhead. Based on this extensive analysis 70 different architectures are proposed in different cases. Note that, fine-grained differentiation between different architectures is an essential part of the analysis. Since the concept of architecture used in the thesis is formalized, one difference in an architecture property results in two different architectures. Although, this leads to a large overall number of proposed architectures, these are mostly similar and are proposed for 4 different problems in several different use-cases.

Since architectures are proposed based on a theoretical methodology, it was also important to demonstrate that the proposed concepts are valid and possible to implement. The thesis describes the implementation of 15 of the proposed architectures. Contributions of this thesis are aiming to ease the work of scientists and help them to exploit the potential of workflows and Grids.

Section 1.4.1 and 1.5 gave a brief introduction of existing solutions, further details of these and other existing solutions are provided in the following chapters, in section 2.4, 3.4, 4.4, and 5.4.

1.6.1 Research method

Based on the existing solutions and how they can be improved by exploiting the capabilities of the Grid, several architectural aspects were considered. The architectural aspects in each case were defined in such way that by combining them it is possible to construct the architectures of the described existing solutions as well as to define more efficient new architectures, that utilise computational and storage machines to distribute the load of the different software components. Properties of the existing and proposed architectures were compared. A mathematical model

was defined to analyse the performance characteristics of the different architectures. The proposed architectures were improved in several properties (such as generality, extendibility, scalability, and overhead) comparing to the architectures of existing solutions as it is described in section 2.4.3, 3.4.3, 4.4.3 and 5.4.3.

In the case of all contributions the stated requirements are defined to cover the most common user scenarios. This work identifies architectures which are optimal from the stated requirements point of view and defines a general analysis method and a mathematical model for comparing the different architectures.

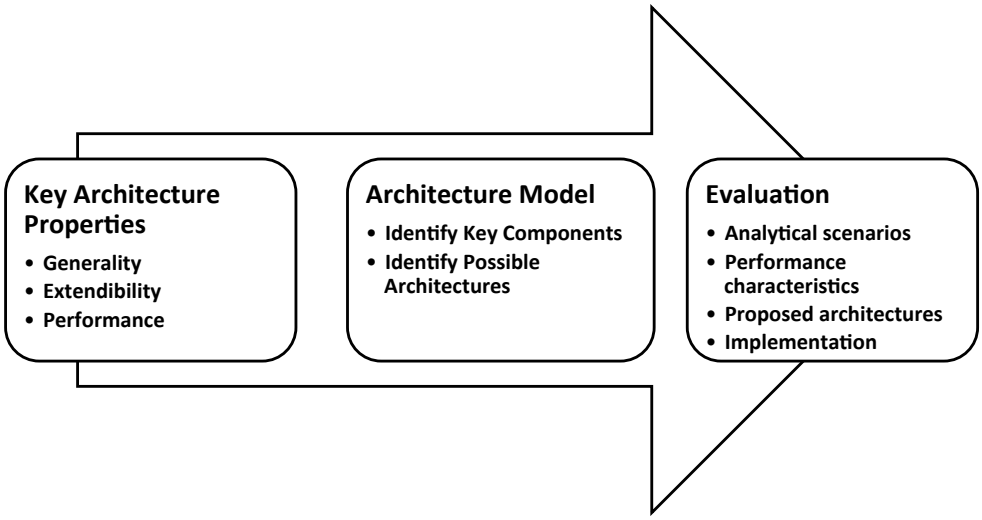

Figure 1.4: Research process

The same research process is applied in the case of each contribution. Key phases of this research process are illustrated in figure 1.4. First, the key architectural properties are identified. Next, the key software components of the model are specified. Based on how these can be distributed over the network and how they can exchange data, the set of possible architectures is constructed. Performance characteristics of these architectures are identified and a set of architectures is proposed.

Architectures are defined in a formal way. Their definition includes structure,

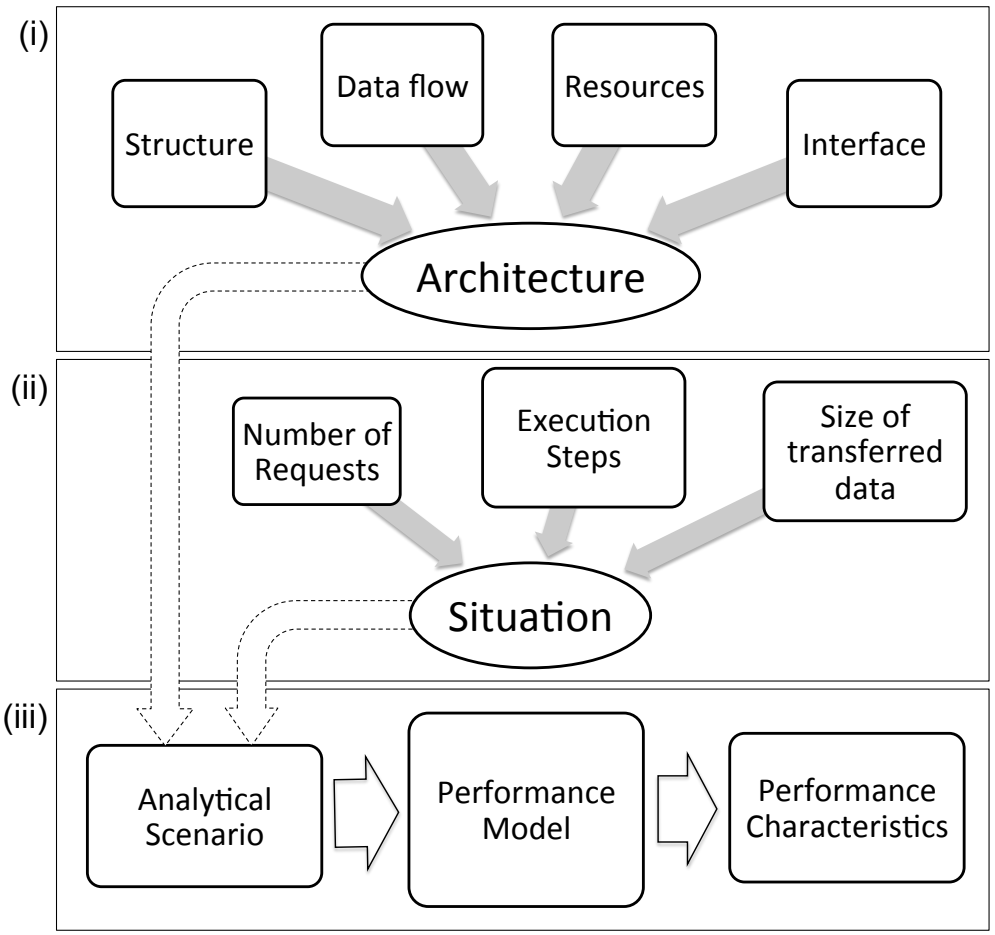

Figure 1.5: Analysis and evaluation, where (i) describes the key components of an architecture, (ii) illustrates the key properties of a situation used for evaluation and (iii) shows the architecture evaluation process.

data flow, resources and interface. These are illustrated in figure 1.5/i. Structure defines the involved software components and their distribution over the network; data flow defines how software components exchange data to fulfil a particular request; resources defines on what kind of machines these components are hosted; and interface defines the key interfaces used to exchange data with external components. Taking into consideration all combinations of these properties, the set of possible architectures is composed. Architectures are analysed based on specific sets of an-

alytical scenarios. An analytical scenario represents a particular architecture in a particular situation, which defines the number of simultaneous requests, the size of transferred data, and the execution steps of the scenario. See illustration in figure 1.5/ii. An analytical scenario serves as an input for the performance model, that maps the performance formulas (request execution time, overhead, latency, scalability) to each architecture in a particular situation. This theoretical approach enables the analysis and evaluation of a large number of architectures, where architectures can be put to maximum load; performance properties can be analysed for arbitrarily large inputs; and evaluation is simple enough to perform within the scope of this thesis.

1.6.2 Contributions

Accessing heterogeneous data resources

The first part of the thesis focuses on access to heterogeneous data provided for Grid applications and Workflows. It attempts to address the following research questions:

1. How data access should be provided for large numbers of applications running on single computational machines provided by the Grid?
2. How data access should be provided for large numbers of running workflows?
3. How large amounts of bulk data should be transferred with optimal performance?

These lead to the following contributions:

- **C1:** A set of optimal architectures to realise heterogeneous Data Access Solutions for Grid applications (DASG).

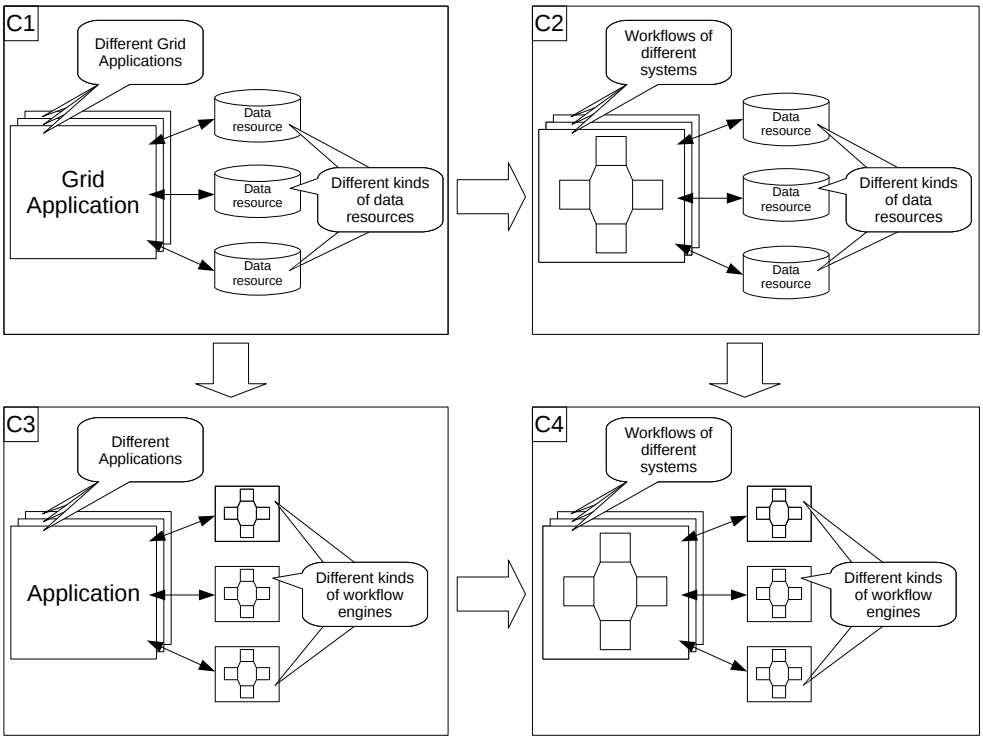

Figure 1.6: Contributions and their relations

- **C2:** A set of optimal architectures to realise heterogeneous Data Access Solutions for Workflows (DASW). This research is based on the results of DASGs (see figure 1.6). However, in this case data access is provided not for Grid applications, but for workflows of different workflow systems.

Executing heterogeneous workflows

The second part of the thesis focuses on workflow interoperability at the level of workflow engines and addresses the following research questions:

1. How can large numbers of applications running on single machines execute workflows of different kinds?

2. How different workflow systems should be connected and how the communication should be established?

3. How can large numbers of running workflow instances invoke each other and exchange large amounts of bulk data?

These lead to the following contributions:

- **C3:** A set of optimal architectures to realise heterogeneous Workflow Execution Solutions for Applications (WESA). This contribution is partially based on C1 (see figure 1.6). In this case however, the architecture integrates and provides access to heterogeneous workflow engines rather than to different data resources and the access is provided for applications in general, not for Grid applications.

- **C4:** A set of optimal architectures to realise heterogeneous Workflow Execution Solutions for Workflows (WESW) - workflow nesting. This contribution is partially based on C2 and C3 (see figure 1.6). In contrast to C2 the resources to which the access is provided are workflow engines. In contrast to C3 the applications to which the heterogeneous engine access is provided are workflows of different workflow systems.

In the case of each contribution research outcomes are: (a) definition of the key properties, architectural aspects, and requirements; (b) definition of a mathematical model that identifies how the different properties, aspects, and requirements are related; (c) analysis of the existing solutions based on the model; (d) proposal of a set of architectures which are optimal from the requirements point of view; (e) reference implementation of selected proposed architectures.

1.6.3 Thesis structure

Chapter 2, 3, 4, and 5 respectively describe the research work carried out in the field on DASGs, DASWs, WESAs, and WESWs. Each of these chapters are structured as follows. First, the key architecture properties of the aimed solutions are specified and the set of possible architectures are defined based on a mathematical model. Using this model, characteristics of the different architectures are identified. Based on this, proposed architectures are selected and compared to the architectures of existing solutions. Finally, reference implementations of selected proposed architectures are described and limitations are identified.

Chapter 2

Heterogeneous Data Access Solutions for Grid applications (DASG)

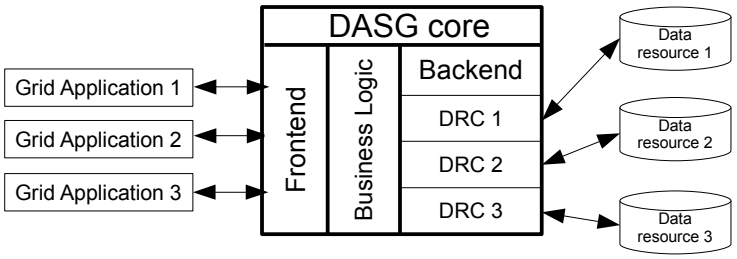

Figure 2.1: Concept of existing DASGs

A DASG typically has a frontend interface, a business logic layer, and a backend, that encapsulates a set of Data Resource Clients (DRC) which are software components that provide access to remote data resources and are able to communicate with them directly. (For example: MySQL client, Oracle client, GridFTP client, SRB client) The application passes a request to the frontend. The business logic

layer defines how the appropriate DRC is selected and executed. The selected DRC sends the request to a given data resource and receives the response, that is transferred back to the application. See illustration in figure 2.1 A few DASGs have been developed in the past few years, such OGSA-DAI [57, 58], GRelC [90, 91, 92, 93], and AMGA metadata catalogue [94, 95, 96]. (These are detailed and analysed in section 2.4.)

2.1 Key DASG properties and requirements

Five key properties of DASGs were identified. These are: generality, extendibility, overhead, latency, and scalability.

Generality As described in section 1.3, numerous heterogeneous data resources coexist and are used by different scientific communities. Therefore, *generality* is a key property of a DASG. An ideal DASG is able to provide access to relational databases, XML databases, file systems, and repositories.

Extendibility Different scientific communities have different scientific problems. They use different Grids, different applications and different data resources. Furthermore, the evolution of Grids and Grid based systems is rapid; new architectures, solutions and standards emerge continuously. Hence, *extendibility* of DASGs is a key property. An ideal DASG can be extended with the support of a new kind of data resource without requiring great effort.

Overhead, latency, and scalability Many scientific problems that e-Scientists deal with involve large-scale computations and require processing large amounts of data. High performance of the integrated systems is essential. When different systems are integrated and connected in order to work together, the mediator that connects the systems in question may become a bottleneck and decrease the performance of the whole application. *Overhead*, *latency*, and *scalability* are also taken under consideration as key properties. Overhead is considered as the delay in transferring a whole data set via a particular DASG compared to transferring it directly between the data resource and the application. Latency is considered as the time required for the first network packet of the transferred data to reach the application machine via a particular DASG. Scalability is represented by the Bachmann–Landau notation, indicating the growth rates of overhead and latency in function of bulk data amount and number of simultaneous requests. All performance related properties are formally defined in section 2.3.2. (See definition 2.52 and 2.53). An ideal DASG is highly scalable and transfers data with low overhead without significant latency.

It should be noted that most existing DASGs provide additional functionality (such as data transformation, additional security, or performing workflows of data requests) on top of the functionality provided by DRCs. However, in many cases this extra functionality is not necessary, the basic functionality provided by the DRCs is sufficient. This research is not aiming to compare the existing solutions based on the additional functionality they provide. Therefore, functionality is not considered as a key property.

2.2 DASG architecture definition

This section not only defines how DASG architectures are represented, but introduces a general architecture model, that will also be used in the case of all further contributions. Definitions of this general architecture model are marked with asterisks.

A DASG architecture consists of four properties *structure*, *data flow*, *resources*, and *interface*. To identify and study possible approaches to realise DASGs in a distributed environment such as Grid, prospective solutions are investigated based on these four aspects.

Existing DASGs realise such an architecture where DRCs are stored and executed on the same machine that hosts the frontend service and the business logic layer. This approach puts all load upon this computer and may result in this machine becoming a bottleneck. Since existing Grid infrastructures provide a vast range of computational and storage resources, DRCs can be stored in remote storages and executed using remote computational resources to distribute the load of a DASG. To see how the functionality of a DASG can be provided if it is distributed between multiple machines, different entities are identified. These are called nodes in this model and form the basic building blocks of a DASG architecture.

2.2.1 DASG node

Definition 2.1 (Node and node type *)
A *node* is a running computer program or function. A *node type* represents a set of computer programs and/or functions. Let $\mathcal{N}$ represent the set of all nodes and $\mathcal{T}$ represent the set of all node types.

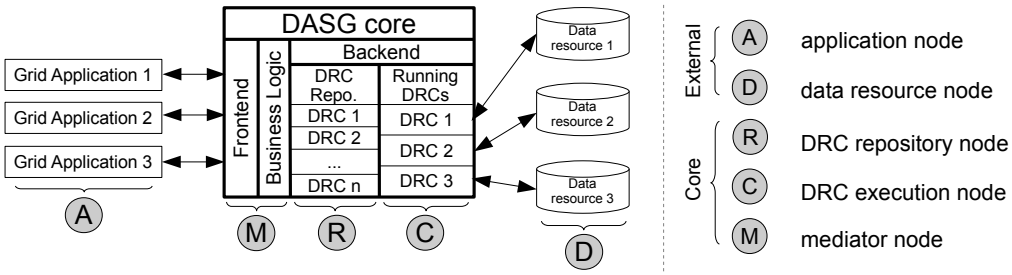

Figure 2.2: Nodes of existing DASGs

Definition 2.2 (DASG node types)

In the case of DASGs the following node types are distinguished:

- An *application node* is a running application that needs to access a data resource. In the case of DASGs this application is always executed on a single computational machine provided by the Grid. Application nodes belong to type A.
- A *data resource node* provides access to a dataset hosted locally. Data resource nodes belong to node type D.
- A *DRC repository node* provides the executable code of a DRC, which is always stored locally to the DRC repository node. Note, that this entity is not necessarily a running service of a digital repository such as (Fedora or DSpace), it can be any entity that is able to provide the executable code of a DRC. DRC repository nodes belong to node type R.
- A *DRC execution node* receives the executable code of a DRC from a DRC repository node and executes it locally. After this point it represents the running DRC. These nodes belong to type C.
- A *mediator node* provides the DASG frontend and the business logic layer. It is contacted to satisfy a particular data request and performs all necessary

steps in order to fulfil this. The mediator is aware of issues such as: the machines that can run the DRC repository nodes; the machines that can run the DRC execution nodes; what DRCs are available in the DRC repositories; etc. Mediator nodes belong to type M.

Based on this let $\mathcal{T} := \{A, M, R, C, D\}$ be the *set of all DASG node types*. Nodes of M, R, C provide a DASG service and enable the communication between the nodes of A and D. Therefore, two disjunctive subsets can be identified within $\mathcal{T}$: let $\mathcal{T}' := \{M, R, C\}$ be the *set of core DASG node types* and let $\mathcal{T}'' := \{A, D\}$ be the *set of external DASG node types*.

In the case of existing DASGs, a mediator node, a DRC repository node, and a DRC execution node are hosted on the same machine. See illustration on figure 2.2.

Definition 2.3 (Instance *)
Exactly one node of each node type is required to perform a request. Therefore, let an *instance* be a set of $|\mathcal{T}|$ nodes, where each node belongs to a different node type of $\mathcal{T}$.

Definition 2.4 (DASG Instance)
Let a *DASG instance* be a set of $|\mathcal{T}| = 5$ nodes, where each node belongs to a different node type of $\{A, M, R, C, D\}$.

Definition 2.5 (Coexistence of multiple instances *)
Each request is performed through data exchanges between the nodes of an instance and each request is performed within a different instance. In order to represent multiple requests, multiple instances are needed. Therefore, the number of instances is equal to the number of requests, which can be any natural number including 0. Let $r \in \mathbb{N}$ be the *number of instances*.

Definition 2.6 (Bijection between node types and instances *)
From definition 2.3, there is exactly one node of each node type within an instance. Therefore, each instance unequivocally defines a bijection between $\mathcal{T}$ and itself. $\forall i \in [1..r]$: let $\varphi_i : \mathcal{T} \rightarrow \mathcal{N}_i$ represent this bijection, where $\forall t \in \mathcal{T} : \varphi_i(t)$ is the ith node that belongs to node type t. This means that to each node type φ_i maps the node which belongs to the given node type.

Definition 2.7 (Bijection between DASG node types and instances)
Let $\forall i \in [1..r]$: let $\mathcal{N}_i := \{A_i, M_i, R_i, C_i, D_i\}$ be the ith DASG instance, where $\varphi_i(A) = A_i$, $\varphi_i(M) = M_i$, $\varphi_i(R) = R_i$, $\varphi_i(C) = C_i$, and $\varphi_i(D) = D_i$.

Definition 2.8 (Node type set *)
$\forall t \in \mathcal{T}$: let $\mathcal{N}_t = \bigcup_{i=1}^{r} \varphi_i(t)$ be *node type set of* t.

In the case of DASGs there are 5 node type sets $\mathcal{N}_A$, $\mathcal{N}_M$, $\mathcal{N}_R$, $\mathcal{N}_E$, $\mathcal{N}_D$ and r nodes in each type set. A DASG node matrix of instances and types can be constructed as illustrated in table 2.1. Furthermore, both $\bigcup_{i=1}^{r} \mathcal{N}_i$ and $\bigcup_{t \in \mathcal{T}} \mathcal{N}_t$ are equal to the set of all DASG nodes, $\mathcal{N}$ and $|\mathcal{N}| = 5r$.

		Node types				
		$\mathcal{N}_A$	$\mathcal{N}_M$	$\mathcal{N}_R$	$\mathcal{N}_C$	$\mathcal{N}_D$
Instances	$\mathcal{N}_1$	A_1	M_1	R_1	C_1	D_1
	$\mathcal{N}_2$	A_2	M_2	R_2	C_2	D_2
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	$\mathcal{N}_r$	A_r	M_r	R_r	C_r	D_r

Table 2.1: DASG node matrix where rows are instances and columns are node type sets.

2.2.2 DASG structure

A structure defines the relationships between the nodes in terms of whether two nodes are running on the same or on different machines.

Definition 2.9 (Coupling *)
Two nodes are *coupled* if they are hosted on the same machine. A node is *decoupled* if it is not coupled with any other node.

Definition 2.10 (Structure *)
A *structure* is an equivalence relation over $\mathcal{N}$ representing which nodes of $\mathcal{N}$ are coupled and which are not. Let $\mathscr{G}_r := \{G \subset \mathcal{N}^2 \| G \text{ is an equivalence relation } \}$ represent the set of all possible structures for r instances.

A structure is specific to the number of instances and necessary for the performance analysis of the different architectures. However, since this number changes with the number of requests, another concept is used for architecture definition to represent the layout independently of r. This concept is called *structure layout* and is defined by two attributes:

- *Instance layout* determines couplings between the nodes of each instance.
- *Type layout* determines couplings between the nodes of each node type.

Definition 2.11 (Instance layout *)
An *instance layout* on domain $\mathcal{T}$ is an equivalence relation on $\mathcal{T}$ that defines for each node type pair whether the nodes that belong to them are or are not coupled within each instance. The set of all possible instance layouts on domain $\mathcal{T}$ is represented by $\mathcal{L}_I(\mathcal{T})$ and can be constructed as:

$$\mathcal{L}_I(\mathcal{T}) := \{h \subset \mathcal{T}^2 \| h \text{ is an equivalence relation}\}. \tag{2.1}$$

Structure $G \in \mathscr{G}_r$ *implements* instance layout h if:

$$\forall i \in [1..r], \forall (t_1, t_2) \in \mathcal{T}^2 : (t_1, t_2) \in h \Leftrightarrow (\varphi_i(t_1), \varphi_i(t_2)) \in G. \tag{2.2}$$

This means that within each instance two nodes are hosted on the same machine based on G if and only if their node types are in the same equivalence class based on h.

Definition 2.12 (DASG instance layout)
The set of all possible DASG instance layouts is represented by $\mathcal{L}_I$ and equals to the set of all possible instance layouts on domain $\{A, M, R, C, D\}$.

A type layout defines for each node type between how many machines its nodes are distributed.

Definition 2.13 (Type layout *)
A *type layout* on domain $\mathcal{T}$ is a function $\delta : \mathcal{T} \to \mathbb{N}^+$, where $\delta(t)$ $(=\delta_t)$ represents the number of available machines that can host the nodes of node type t $(t \in \mathcal{T})$. The set of all possible type layouts on domain $\mathcal{T}$ is represented by $\mathcal{L}_T(\mathcal{T})$ and can be constructed as:

$$\mathcal{L}_T(\mathcal{T}) := [\mathcal{T} \to \mathbb{N}^+] \tag{2.3}$$

A structure $G \in \mathcal{G}_r$ implements a type layout δ if:

$$\forall i, j \in [1..r], t \in \mathcal{T} : i \equiv j \mod \delta_t \Leftrightarrow (\varphi_i(t), \varphi_j(t)) \in G. \tag{2.4}$$

Note that to ensure the fair distribution of nodes between the available machines, within each structure that implements a type layout δ, $\forall t \in \mathcal{T}$ nodes of type t are distributed based on round robin scheduling between the available δ_t machines (this is provided by condition 2.4). Note that any scheduling policy that ensures fair distribution would be suitable.

Definition 2.14 (DASG type layout)

The set of all possible DASG type layouts is represented by $\mathcal{L}_T$ and equals to the set of all possible type layouts on domain $\{A, M, R, C, D\}$.

Definition 2.15 (Structure layout *)

A *structure layout* on domain $\mathcal{T}$ is a couple composed of an instance layout h and a type layout δ which are both defined on domain $\mathcal{T}$ and for any two node types coupled in instance layout h (meaning that within each instance their nodes are hosted on the same machine) their nodes are distributed between the same number of machines based on δ. It can be proven that this condition ensures there always is a structure which implements both h and δ. (For details, see lemma A.1.) The set of all possible structure layouts on domain $\mathcal{T}$ is represented by $\mathcal{L}_S(\mathcal{T})$ and can be constructed as:

$$\mathcal{L}_S(\mathcal{T}) := \{(h, \delta) \in \mathcal{L}_I(\mathcal{T}) \times \mathcal{L}_T(\mathcal{T}) \| \forall (t_1, t_2) \in h : t_1, t_2 \in \mathcal{T} \Rightarrow \delta_{t_1} = \delta_{t_2}\}. \quad (2.5)$$

$G \in \mathscr{G}_r$ implements structure layout (h, δ) if and only if it implements both h and δ. Note that, based on the above, the predicate in the set definition guarantees that there always is such structure. See the nodes of M and R in example 2.1.

Definition 2.16 (DASG structure layout)

The set of all possible DASG structure layouts is represented by $\mathcal{L}_S$ and equals to the set of all possible structure layouts on domain $\{A, M, R, C, D\}$.

Example (A DASG structure layout)

Let $(h, \delta) \in \mathcal{L}_S$ be a DASG structure layout, where $h = \overline{\{(M, R)\}}$ and $\delta = \{(A, 4)$, $(M, 2)$, $(R, 2)$, $(C, 3)$, $(D, 4)\}$. In order to save space, rather than defining h by listing all couplings it contains, in most cases it is defined as a transitive, reflexive, symmetric closure. Figure 2.3 illustrates a structure that implements structure

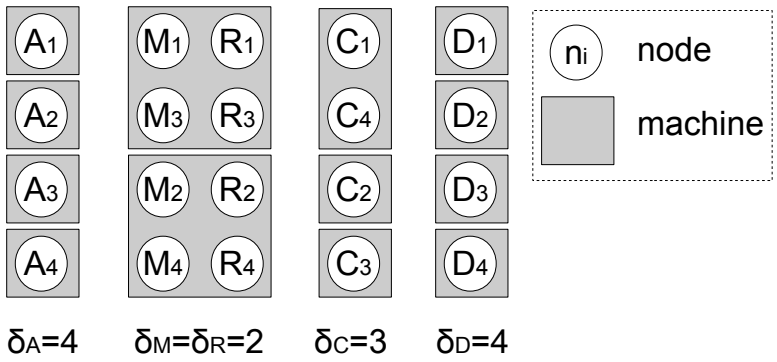

Figure 2.3: DASG structure that implements both instance layout $h = \overline{\{(M, R)\}}$ and type layout $\delta = \{(A, 4), (M, 2), (R, 2), (C, 3), (D, 4)\}$ with 4 simultaneous requests ($r = 4$).

layout (h, δ) for 4 instances ($r = 4$). The figure shows that the illustrated structure is an equivalence relation where nodes are hosted on the same machine if they are in the same equivalence class.

2.2.3 DASG data flow

Definition 2.17 (DASG data types)
Data between distributed nodes of an instance can flow in various ways. To identify the different possibilities, three types of data are distinguished:

- *bulk data* is the dataset that needs to be transferred between the data resource (D) and application nodes (A);
- *DRC data* is the DRC itself that needs to be transferred from the DRC repository nodes (R) to the DRC execution nodes (C); and
- *control data* is the set of information that includes all further data transferred between the nodes. The latter consists of a small number of requests which

are necessary to exchange in order to provide access for an application to a data resource.

The amount of control data is typically measured in kilobytes, whilst DRC data is measured in megabytes, and bulk data in giga- or terabytes. The way bulk data is transferred is critical. In comparison to this, DRC data flow has a marginal impact and can be considered but control flow is insignificant in affecting the performance of a particular DASG architecture. Hence, only two kinds of data flow are considered: DRC flow, and bulk data flow. Definition of both data flows are based on the concept of path layout and data staging.

Definition 2.18 (Path *)

A path defines a sequence of nodes. Let $\mathcal{P}$ represent the set of all possible paths, where:

$$\mathcal{P} := \{(n_1, n_2, ..., n_m) \in \mathcal{N}^m \| m \in \mathbb{N}^+\}. \tag{2.6}$$

$\forall i \in [1..m]$ the ith node of the path is n_i and the *source* and *destination* nodes of the path are always n_1 and n_m respectively. Path p is *acyclic* based on structure $G \in \mathscr{G}_r$ if it satisfies the following condition:

$$\forall i, j \in [1..m] : (n_i, n_j) \in G \Leftrightarrow \forall k \in [i..j] : (n_i, n_k) \in G. \tag{2.7}$$

Definition 2.19 (Path layout *)

A *path layout* defines a sequence of node types. Let $\mathcal{L}_P(\mathcal{T})$ represent the set of all path layouts, where:

$$\mathcal{L}_P := \{(t_1, t_2, ..., t_m) \in \mathcal{T}^m \| m \in \mathbb{N}^+\}. \tag{2.8}$$

$\forall i \in [1..m]$ the ith node type of the path layout is t_i and the *source* and *destination* node types of the path layout are always t_1 and t_m respectively. Path layout q is

acyclic based on instance layout $h \in \mathcal{L}_I(\mathcal{T})$ if any two node types in q are coupled based on h then all node types in q between those two are coupled as well. With other words, $q = (t_1, ..., t_m)$ is acyclic based on h if it satisfies the following condition:

$$\forall i, j \in [1..m] : (t_i, t_j) \in h \Rightarrow \forall k \in [i..j] : (t_i, t_k) \in h. \tag{2.9}$$

Definition 2.20 (DASG DRC path layout)
Two DRC path layouts are distinguished in the case of DASGs: when the DRC is transferred directly (R, C) and when it is transferred via the mediator (R, M, C). DASG DRC path layouts that involve external nodes are not considered. Based on these, let $\mathcal{D}_P := \{(R, C), (R, M, C)\}$ be the *set of DRC path layouts*.

Note, that DRC is always transferred from R to C, whereas bulk data can flow in both directions: from D to A or from A to D. From transfer time and overhead point of view the direction of transfer does not make any difference, hence in the case of bulk data only direction D to A is considered.

Definition 2.21 (DASG bulk data path types)
From D bulk data always have to be transferred to C, since this represents the running DRC which is the only entity that can communicate with D. Next bulk data is either transferred to the application directly or via the mediator. Cases that transfer bulk data via the DRC repository (R) are not considered. Based on these, let $\mathcal{B}_P := \{(D, C, A), (D, C, M, A)\}$ be the set of bulk data paths.

Definition 2.22 (Mapping between path layouts and paths *)
A path layout defines a path over the nodes of each instance. $\forall i \in [1..r]$: let $\psi_i : \mathcal{L}_P(\mathcal{T}) \rightarrow \mathcal{P}$ map this path to a path layout as $\psi_i((t_1, ..., t_m)) := (\varphi_i(t_1), ..., \varphi_i(t_m))$.

In order to define data staging, the following concepts need to be introduced first.

Definition 2.23 (Byte array and its length *)

A *byte array of length* l is an element of $\mathbb{B}^l$. Let $\mathcal{B} = \bigcup_{i=0}^{\infty} \mathbb{B}^i$ be the set of all byte arrays, let $\lambda : \mathcal{B} \to \mathbb{N}$ represent the *length of a byte array* ($\lambda(x) = l \Leftrightarrow x \in \mathbb{B}^l$), and $\forall x \in \mathcal{B}, i \in [1..\lambda(x)]$ let $x[i]$ represent the ith byte of a byte array.

Definition 2.24 (Byte array concatenation *)

Let $x_1x_2...x_k = x \in \mathcal{B}$ be the *concatenation* of byte arrays $x_1, x_2, ..., x_k \in \mathcal{B}$, where $\forall j \in [1..k], i \in [1..\lambda(x_j)] : x[\lambda(x_1x_2...x_{j-1}) + i] = x_j[i]$.

Definition 2.25 (Transferring a byte array via a path *)

Assuming that $p = (n_1, ..., n_m) \in \mathcal{P}$, $x \in \mathcal{B}$ is transferred via p as follows:

1. x is transferred from n_1 to n_2.
2. $\forall j \in [3..m] : x$ is transferred from n_{j-1} to n_j as soon as transfer of x from n_{j-2} to n_{j-1} has finished.

Definition 2.26 (Transferring a sequence of byte arrays via a path *)

Assuming that $p = (n_1, ..., n_m) \in \mathcal{P}$, $(x_1, x_2, ..., x_k), (k \in \mathbb{N}^+)$ sequence of byte arrays is transferred via p as follows:

1. x_1 is transferred from n_1 to n_2.
2. $\forall j \in [3..m] : x_1$ is transferred from n_{j-1} to n_j as soon as transfer of x_1 from n_{j-2} to n_{j-1} has finished.
3. $\forall i \in [2..k] : x_i$ is transferred from n_1 to n_2 as soon as transfer of x_{k-1} from n_1 to n_2 has finished.

4. $\forall i \in [2..k], j \in [3..m] : x_i$ is transferred from n_{j-1} to n_j as soon as transfer of x_i from n_{j-2} to n_{j-1} has finished and transfer of x_{i-1} from n_{j-1} to n_j has finished.

Definition 2.27 (Pipelined transfer *)
Let $x \in \mathcal{B}$, $p \in \mathcal{P}$, $s \in \mathbb{N}^+$, where $\lambda(x)$ is dividable by s. *Pipelined transfer* of byte array x via path p with slice size s means that a sequence of byte arrays $(x_1, x_2, ..., x_k)$ is generated from x, where $k = \frac{\lambda(x)}{s}$ and $\forall i \in [1..k] : \lambda(x_i) = s$, $x_1x_2...x_k = x$ and this sequence of byte arrays is transferred via path p according to definition 2.26.

Note that in order to simplify the model, in the case of pipelined transfer the generated byte array sequence contains only byte arrays of equal length and $\lambda(x)$ have to be dividable by s.

Definition 2.28 (Non pipelined transfer *)
Non pipelined transfer of a byte array via a path means that the byte array is transferred via a path according to definition 2.25.

Definition 2.29 (DASG DRC staging)
Let $\mathcal{D}_S = \{Pip, \neg Pip\}$ be the *set of DRC staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined DRC staging.

Definition 2.30 (DASG bulk data staging)
Let $\mathcal{B}_S = \{Pip, \neg Pip\}$ be the *set of DRC staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined DRC staging.

Definition 2.31 (Set of DASG data flow types)
Let $\mathcal{DF} := \mathcal{D}_P \times \mathcal{D}_S \times \mathcal{B}_P \times \mathcal{B}_S$ be the set of possible DASG data flow types.

2.2.4 DASG resource

It is also important, on what machine resources the different nodes are hosted. This aspect is defined based on the following concepts.

Definition 2.32 (Resource types *)
Based on the services a machine provides, this model distinguishes between three basic types of machines: *Computational Machines* (CoM), *Storage Machines* (StM), and *Dedicated Machines* (DeM). A computational machine provides services for executing, monitoring, and cancelling jobs. A storage machine provides services for uploading, storing, locating, and downloading any kind of data. A dedicated machine, can run any service without restriction in order to fulfil a specific purpose.

	Dedicated machines (DeM)	Computational machines (CoM)	Storage machines (StM)
Advantages	- For hosting any custom service - No dependency problems - No firewall problems - Can host any operating system	- Special off-the-shelf services - Large numbers of machines - Machines are maintained - Services are maintained - No cost	- Special off-the-shelf services - Large numbers of machines - Machines are maintained - Services are maintained - No cost
Disadvantages	- Machine has to be provided - Machine has to be maintained - Service has to be maintained - Cost demanding	- Not designed for providing custom services - Dependency problems - Firewall problems - Only for running applications - Given operating system - Job queues	- Not designed for providing custom services - Cannot run applications - Only for data

Table 2.2: Properties of different resource types

Table 2.2 summarises the properties of these resource types. The main difference between dedicated machines and the other two resource types is that dedicated machines provide great flexibility, but the machine and its services have to be provided and maintained. Therefore, hosting nodes on dedicated machines is expensive, especially in a large scale. Computational and storage machines are more restrictive, they provide a fix set of services, but these are maintained by existing Grid

infrastructures. Hence, they can be utilised as free-off-charge, off-the-shelf services by the academic/scientific community. The machines that host external nodes are considered unknown. A fourth resource type called *External Machines* (ExM) is used to represent machines of this kind. *Resource type* of a node is the resource type of the machine that hosts the node. Let $\mathcal{H}_R := \{CoM, StM, DeM, ExM\}$ be the *set of all resource types*.

Definition 2.33 (Resource layout *)
A *resource layout* on domain $\mathcal{T}$ is a $\xi : \mathcal{T} \rightarrow \mathcal{H}_R$ function, where $\xi(t)$ ($=\xi_t$) represents the resource types of the machines that host the nodes of node type t ($t \in \mathcal{T}$). Let $\mathcal{L}_R(\mathcal{T})$ represent the set of all resource layouts on domain $\mathcal{T}$, where:

$$\mathcal{L}_R(\mathcal{T}) := [\mathcal{T} \rightarrow \mathcal{H}_R] \tag{2.10}$$

Definition 2.34 (DASG resource layout)
Based on definition 2.2, mediator nodes have to provide custom services which are not available on computational or storage machines. Therefore, mediator nodes always have to be hosted either on dedicated or on external machines. Repository nodes have to provide the executable code of the DRCs. This functionality can be provided by the services of the storage machines, by custom services hosted on dedicated machines, or by services running on external machines, but cannot be hosted by computational machines, since in most cases it is not possible to store data on those machines at all. (Note that although in some cases it is possible to store data temporarily on computational machines, there is no guarantee the data will reside there.) DRC execution nodes can run on either computational machines, on dedicated machines, or on external machines, storage resources cannot run any application. *Resource layout* defines for all DASG node types that what types of machines their nodes are hosted. Therefore, based on the above, the set of all

possible DASG resource layouts is defined over node type set $\mathcal{T}$ as:

$$\mathcal{L}_R := \{\xi \in \mathcal{L}_R(\mathcal{T}) \| \xi_M \in \{DeM, ExM\} \wedge \xi_R \neq CoM \wedge \xi_C \neq StM \wedge \\ \wedge \xi_A, \xi_D = ExM\} \tag{2.11}$$

2.2.5 DASG interface

Different interfaces connect different nodes of $\mathcal{N}$. These interfaces determine how nodes can interact and exchange data. Interfaces can be analysed based on several aspects. This model focuses on two of these aspects: how an interface is represented and how general an interface is.

Definition 2.35 (Interface representation *)
An interface can be represented as a *Command Line Interface* (CLI) or as an *Application Programming Interface* (API).

The basic difference between the two is that CLIs can be accessed by both humans and software, while APIs can be accessed only by software.

Definition 2.36 (Interface generality *)
An interface can be *specific* (Spe) to certain types of requests if the number or types of input and output parameters are restricted. It can be *generic* (Gen) if the number and type of input and output parameters of the requests are not limited by the interface, hence, it does not restrict the set of data exchanges that can be performed via it.

Definition 2.37 (DASG frontend interface)
Frontend interface ($\mathcal{I}_F$) is the interface through which applications can utilize the provided functionality of a DASG. Let $\mathcal{I}_A := \{Gen, Spe\}$ be the set of application interface types. (See definition 2.36.)

Representation of $\mathcal{I}_F$ (see definition 2.35) is not considered as part of a DASG architecture, because a mapping of a single frontend interface to another interface representation type (i.e. CLI to API) is rather straightforward to realise. However, generality is vital, since it determines the set of data requests that can be performed via DASG.

Definition 2.38 (DASG backend interface)
Backend interface ($\mathcal{I}_B$) defines how DRCs can be accessed. Let $\mathcal{I}_B := \{CLI, API\}$ be the set of backend interface types.

Since DRCs are designed to access a particular data resource, DRC interface is always specific to a data resource. However, in terms of backend interface, representation is vital, since it determines how an existing DASG can be extended with the support of further DRCs. Most vendors provide both API and CLI representations for accessing their data resources.

Definition 2.39 (Set of possible DASG interfaces)
The set of possible interfaces can be defined as $\mathcal{IN} := \mathcal{I}_F \times \mathcal{I}_B$.

2.2.6 DASG architecture and solution

Definition 2.40 (Set of possible DASG architectures)
A *DASG architecture* defines structure layout, resource layout, data flow, and interfaces. Let $\mathcal{AR}$ represent the set of all DASG architectures, where:

$$\begin{aligned}\mathcal{AR} := \{((h,\delta),\xi,(q_d,g_d,q_b,g_b),(i_f,i_b)) \in \mathcal{L}_S \times \mathcal{L}_R \times \mathcal{DF} \times \mathcal{IN} \| \\ q_d \text{ and } q_b \text{ are acyclic path layouts based on instance layout } h \wedge \quad (i) \\ \wedge \forall (t,D) \in h : t = D \wedge \quad (ii) \\ \wedge \forall (t_1,t_2) \in h : \xi_{t_1} = \xi_{t_2}\}. \quad (iii)\end{aligned} \tag{2.12}$$

Note that condition (i) eliminates DASG architectures that transfer data in loops between machines, since in most cases this is pointless. The purpose of a DASG is to provide access to remote data resources. Hence, condition (ii) ensures that data resources are always remote, not coupled with any other node. Condition (iii) ensures that if two nodes are hosted on the same machine, then they have the same resource type.

Definition 2.41 (DASG solution)
A *DASG solution* is a set of DASG architectures. With other words, it is a not empty subset of $\mathcal{AR}$.

2.3 DASG architecture analysis

2.3.1 DASG generality and extendibility

Generality of a DASG architecture (types of data requests that can be executed and the set of data resources that can be connected via it) depends on the frontend interface. By applying a specific frontend interface the usage of the solution can be simplified, but this also restricts the provided functionality. In order to provide access to a wider set of data resources without any restrictions on the type and number of input and output parameters frontend interface should be general.

Extendibility of an architecture is determined by how easy it is to extend the set of available DRCs. In terms of backend interface, CLI is recommended, since it enables the straightforward extension and update of the set of supported DRCs without requiring programming skills. On the other hand, the mediator knows about the available DRC repositories and the available DRCs. If the system is extended with a new workflow engine, the mediator has to be updated. Note that

the mediator has to be aware of all engines and all engine repositories. Adding a new engine does not only imply that the mediator has to be updated, it might require changes in the business logic of the mediator as well. In the case of instance layouts that have (A, M) coupled, each application has a copy of the mediator, in which case each application has to be updated with the new mediator version. However, if the mediator runs as a centralized service, only one update has to be performed. Therefore, it is recommended to have a centralised mediator which is not coupled with the application and is hosted on a dedicated machine.

2.3.2 DASG performance

The aim of the performance analysis is to compare overhead, latency, and scalability of different DASG architectures and show how these values vary with bulk data volume and number of simultaneous requests. The performance comparison is based on DASG scenarios where $r \in \mathbb{N}^+$ applications hosted by r different machines gather data of equal size from r decoupled data resources via a particular DASG architecture simultaneously. These scenarios are represented as elements of the set defined below.

Definition 2.42 (DASG scenarios)

$$
\begin{aligned}
\text{Let } \mathcal{AS} := \{((h,\delta),\xi,q_d,q_b,w_d,s_d,s_b,l_d,l_b,r) \in \mathcal{L}_S \times \mathcal{L}_R \times \mathcal{D}_P \times \mathcal{B}_P \times \mathbb{R}_0^+ \times (\mathbb{N}^+)^5 \| \\
\forall t \in \mathcal{T}'' : \delta_t = r \wedge & \quad (i) \\
\wedge q_d \text{ and } q_b \text{ are acyclic path layouts based on } h \wedge & \quad (ii) \\
\wedge \forall (t,D) \in h : t = D \wedge & \quad (iii) \\
\wedge \forall (t_1,t_2) \in h : \xi_{t_1} = \xi_{t_2} \wedge & \quad (iv) \\
\wedge \xi_C \neq CoM \Rightarrow w_d = 0 \wedge & \quad (v) \\
\wedge \forall t \in \mathcal{T}' : r \equiv 0 \mod \delta_t \wedge & \quad (vi) \\
\wedge l_d \equiv 0 \mod s_d \wedge l_b \equiv 0 \mod s_b \}. & \quad (vii)
\end{aligned}
\tag{2.13}
$$

be the *set of analytical DASG scenarios*. Let $a = ((h,\delta),\xi,q_d,q_b,w_d,s_d,s_b,l_d,l_b,r) \in \mathcal{AS}$ be an analytical scenario. Condition (i) ensures that none of the application nodes nor the data resource nodes are coupled, all are hosted on different machines. a determines structure layout, resource layout, and data flow of a DASG architecture, this is ensured by conditions (ii), (iii), and (iv).

By (h,δ) and ξ, a explicitly defines a DASG structure and resource layout. In terms of *data flow*, a defines DRC and bulk data path layout explicitly by q_d and q_b. DRC and bulk data size are represented by l_d and l_b, DRC and bulk data slice size by s_d and s_b. These implicitly define DRC and bulk data staging as $k_d := \frac{l_d}{s_d}$ and $k_b := \frac{l_b}{s_b}$, where staging is non pipelined if slice number equals to 1 and pipelined otherwise. Note that condition (vii) ensures that l_d is dividable by s_d and l_b is dividable by s_b. w_d represents the amount of time that elapses between the DRC transfer is finished and its execution is started. This delay is resulted by the DRC waiting in job queue if it is executed on a computational node. To simplify the model, this number is the same for each request and it also can be 0 representing cases where the job queues are empty or there are no queues at all. Condition (v)

ensures that w_d is always 0 if the DRC is not executed on a computational machine.

Having conditions (vi) and (vii) the analysis can be simplified. In particular, condition (vi) ensures that all nodes of any core DASG node type is coupled with equal number of nodes of that particular type. This ensures that nodes of each node type can be equally distributed between the available machines. As it was already mentioned condition (vii) ensures that DRC and bulk data size is always dividable by DRC and bulk data slice size respectively, meaning that each slice is of the same size. Although these conditions do not allow to specify scenarios for any r, l_d, s_d, l_b, s_b, w_d, these values can be arbitrarily large. Therefore, based on DASG scenarios the performance characteristics of any DASG architecture can be analysed and compared for arbitrarily large r, l_d, s_d, l_b, s_b, w_d.

Definition 2.43 (DASG scenario execution)
Since control flow is excluded from the model (see definition 2.17), the analysis is based on DRC and bulk data flow. $\forall i \in [1..r]$: let $d_i \in \mathcal{B}$ byte array represent the DRC that communicates with D_i and to be transferred from R_i to C_i and $b_i \in \mathcal{B}$ byte array represent the bulk data that is to be transferred from D_i to A_i. A DASG scenario is executed in three steps:

1. DRC transfer: $\forall i \in [1..r]$: d_i is transferred from R_i to C_i via path $\psi_i(q_d)$ simultaneously.

2. DRC queuing: all DRCs are waiting w_d amount of time to be scheduled for execution,

3. Bulk data transfer: $\forall i \in [1..r]$: the execution of the ith DRC starts and b_i is transferred from D_i to A_i via path $\psi_i(q_b)$ simultaneously.

In order to define the performance characteristics a DASG architecture, the performance characteristics of the above data transfers need to be specified. In

order to do this, first the properties of the network has to be identified. Machines that host the different entities of an architecture are geographically distributed and connected through the existing infrastructure provided by the internet. Since it is not aimed by this research to analyse performance of different network environments and computers, the model is based on a homogeneous computer network, where it is assumed that all machines have the same specification and machines are connected via a constant bandwidth full-duplex network, allowing machines to send and receive data simultaneously without performance decrease. This is defined by the following axioms:

1. Full-duplex network adapters: each machine has an input and an output network port.
2. Fully connected network: the input port of each machine is connected with the output port of each machine.
3. Constant bandwidth: the input port of each machine can receive and the output port of each machine can send constant amount of data per unit time.

These axioms imply several lemmas to define the performance characteristics of the different architectures. In order to define transfer time between two nodes, the concept of port functions are introduced.

Definition 2.44 (Port functions *)
Based on axiom 1, each machine has an input and an output port. Let $\varrho_{ip}, \varrho_{op} : \mathcal{N} \rightarrow \mathbb{N}$, where $\varrho_{ip}(n)$, $\varrho_{op}(n)$ $(n \in \mathcal{N})$ are the number of byte arrays that are transferred simultaneously via the input and output ports (respectively) of the machine that hosts node n. It is assumed that these values are constant for each node.

Based on the axioms, transfer time of a byte array between two non-coupled nodes is determined by its length, the bandwidth, and the number of byte arrays which are

concurrently transferred to/from the host machines of the two nodes. Bandwidth is represented by constant $K \in \mathbb{R}^+$ which is a property of the homogeneous network. Let $\mathcal{N}$ be the set of nodes, let $n, m \in \mathcal{N}$, $x \in \mathcal{B}$ be a byte array of length $l = \lambda(x)$ that is transferred from n to m, and let $G \in \mathscr{G}_r$ be a structure. The following four data transfer cases are distinguished.

- If n and m are hosted on the same machine, it is not considered how x is exchanged between the nodes, transfer time of x is always 0. See example (i) in figure 2.4.
- Transfer time of x is linear with l if n and m are not coupled and other byte arrays are not transferred from the machine that hosts n and to the machine that hosts m. In this case, $\varrho_{op}(n) = 1$, $\varrho_{ip}(m) = 1$, and transfer time of x is Kl. See example (ii) in figure 2.4.
- Transfer time of x is linear with both l and r if n and m are not coupled, and $r - 1$ other byte arrays are transferred from the host machine of n and $r - 1$ further byte arrays are transferred to the host machine of m simultaneously. In this case $\varrho_{op}(n) = r$, $\varrho_{ip}(m) = r$, and transfer time of x is rKl. See example (iii) in figure 2.4.
- If n and m are not coupled, and $\varrho_{op}(n) \neq \varrho_{ip}(m)$, then transfer time of x is determined by the machine that bottlenecks the transfer. In this case, transfer time of x is $\max\{\varrho_{op}(n), \varrho_{ip}(m)\}Kl$. See example (iv) in figure 2.4.

Definition 2.45 (Time of byte array transfer between two nodes *)
Based on the above four cases, let $\tau_e : \mathbb{N} \times \mathcal{N}^2 \to \mathbb{R}_0^+$ be the time that is required for transferring a byte array of a given length between two nodes, where:

$$\tau_e(\lambda(x), (n, m)) := \chi((n, m) \notin G) \max\{\varrho_{op}(n), \varrho_{ip}(m)\}K\lambda(x). \tag{2.14}$$

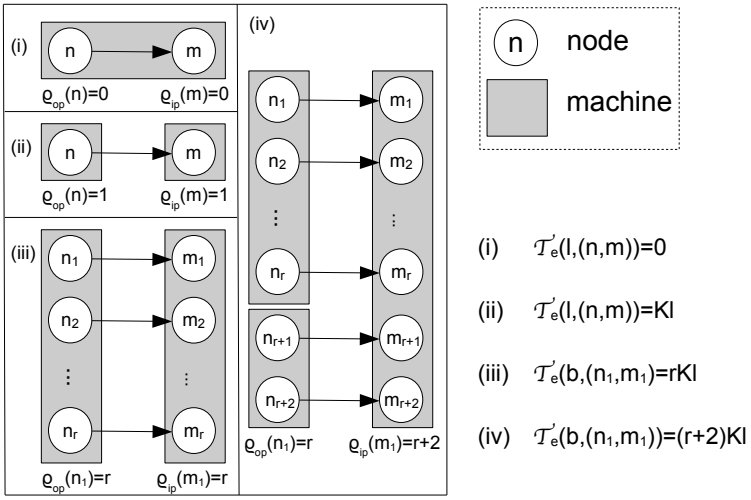

Figure 2.4: Examples for each data transfer case

The formula above is used to calculate the transfer time of a byte array between two nodes. The following describes how transfer time is calculated over a path of nodes. Let $p = (n_0, n_1, ..., n_m) \in \mathcal{P}$ a path over $\mathcal{N}$ and byte array x is transferred via path p, where $k \in \mathbb{N}^+$, $x = x_1x_2...x_k$, $\forall i \in [1..k] : \lambda(x_i) = s$, and $\lambda(x)$ is dividable by s.

Lemma 2.1 (Time of transferring a byte array sequence via a path *)
By applying mathematical induction based on definition 2.26, it can be proven that time of transferring byte array sequence $(x_1, x_2, ..., x_k)$ via path p equals to:

$$\sum_{j=1}^{m} \tau_e(s, (n_{j-1}, n_j)) + (k-1) \max_{j=1}^{m} \tau_e(s, (n_{j-1}, n_j)). \tag{2.15}$$

Proof is provided in appendix A in lemma A.2.

Definition 2.46 (Time and latency of pipelined transfer *)
Let $\tau_p : \mathbb{N} \times \mathbb{N}^+ \times \mathcal{P} \to \mathbb{R}_0^+$ be the time of pipelined transfer of a byte array via a path, where the first parameter represents the length of the byte array, the second parameter represents the slice size, and the last parameter represents the path via which the slices are transferred. Time of pipelined transfer of byte array x via path

p with slice size s equals to:

$$\tau_p(\lambda(x), s, p) := \sum_{j=1}^{m} \tau_e(s, (n_{j-1}, n_j)) + (k-1) \max_{j=1}^{m} \tau_e(s, (n_{j-1}, n_j)). \tag{2.16}$$

Latency is calculated according to the following formula:

$$\varepsilon_p(\lambda(x), s, p) := \sum_{j=1}^{m-1} \tau_e(s, (n_{j-1}, n_j)). \tag{2.17}$$

The above definition is based on a lemma A.2 and definition 2.27. Byte array sequence $(x_1, x_2, ..., x_k)$ is generated from x with slice size s according to definition 2.27. Time of transferring this k element byte array sequence via path $p = (n_0, n_1, ..., n_m)$ is calculated according to lemma A.2. Latency of a data transfer is considered as the time required for the first network packet of the transferred data to reach the destination node. Hence, it is the time required for the first slice of x to reach the n_{m-1} plus the time that is required to transfer the first network packet of the first slice from n_{m-1} to n_m. To simplify the model, the latter is not considered.

Definition 2.47 (Time and latency of non pipelined transfer *)
Latency and time of non pipelined transfer of byte array x via path p equals to: $\varepsilon_p(\lambda(x), \lambda(x), p)$ and $\tau_p(\lambda(x), \lambda(x), p)$ respectively.

Since definition 2.25 and 2.26 define the same transfer steps in the case when a one element byte sequence is transferred via a path, it can be proven that pipelined transfer of a one element byte array sequence equals to the non pipelined transfer of the same byte array. Therefore, to define time and latency of a non pipelined transfer, definition 2.46 can be applied with slice size $\lambda(x)$, which ensures that x will be transferred as a one element byte sequence.

Lemma 2.2 (Slice size independence *)
If $x \in \mathcal{B}$, $p \in \mathcal{P}$, and path p has only two elements, then $\forall s_1, s_2 \in \mathbb{N}^+$, where $\lambda(x)$ is dividable by s_1, s_2:

$$\tau_p(\lambda(x), s_1, p) = \tau_p(\lambda(x), s_2, p). \tag{2.18}$$

Proof is provided in lemma A.3. Note that this also implies that if p has only two elements, then times of pipelined and non pipelined transfer are equal. Therefore, in this case data staging type is irrelevant from transfer time point of view.

Although, the above definitions and lemmas provide formulas for determining time and latency of a transfer, they are based on τ_e. In order to determine τ_e, the structure of the nodes and the number of concurrent data transfers have to be known. Rather than defining a general model that considers any possible concurrent data transfer cases, this model focuses only on the scenarios that are used to compare different architectures.

Definition 2.48 (Simultaneous transfer via a path layout *)

$$\begin{aligned}\text{Let } \mathcal{D}_{st} := \{((h,\delta),q,s,l,r) \in \mathcal{L}_S(\mathcal{T}) \times \mathcal{L}_P(\mathcal{T}) \times (\mathbb{N}^+)^3 \| \\ q \text{ is an acyclic path layout based on instance layout } h \wedge \quad (i) \\ \wedge l \equiv 0 \mod s \wedge \quad (ii) \\ \wedge \forall t \in \mathcal{T} : r \equiv 0 \mod \delta_t\} \quad (iii)\end{aligned} \tag{2.19}$$

be the *set of simultaneous data transfers*, where (h,δ) represents a structure layout, q represents a path layout, s represents slice size, l represents byte array length, and r represents the number of simultaneous requests. Condition (i) ensures that data is never transferred in loops between machines, condition (ii) ensures that l is dividable by s, and condition (iii) ensures that $\forall t \in \mathcal{T} : r$ is dividable by δ_t which implies that nodes of each type are equally distributed between the available machines. Simultaneous data transfer $((h,\delta),q,s,l,r)$ represents the parallel transfer of r different byte arrays of length l, where $\forall i \in [1..r]$: the ith byte array is transferred via path $\psi_i(q)$, transfer is pipelined with slice size s.

Lemma 2.3 (Performance characteristics of simultaneous transfer *)
Let $((h,\delta),q,s,l,r) \in \mathcal{D}_{st}$ be a simultaneous transfer. If $\forall x_1,x_2,...,x_r \in \mathcal{B} : \lambda(x_1) = \lambda(x_2) = ... = \lambda(x_r) = l$, $q = (t_0,t_1,...,t_m)$, and $k = \frac{l}{s}$, then $\forall i \in [1..r]$: the pipelined transfer time of x_i through path $\psi_i(q)$ with slice size s is the same and equals to:

$$\begin{aligned}&\sum_{j=1}^{m} \chi((t_{j-1},t_i) \notin h)\frac{rKs}{\min(r,\delta_{t_{j-1}},\delta_{t_j})} + \\ &+ (k-1)\max_{j=1}^{m} \chi((t_{j-1},t_j) \notin h)\frac{rKs}{\min(r,\delta_{t_{j-1}},\delta_{t_j})},\end{aligned} \tag{2.20}$$

while latency is also the same and equals to:

$$\sum_{j=1}^{m-1} \chi((t_{j-1},t_j) \notin h)\frac{rKs}{\min(r,\delta_{t_{j-1}},\delta_{t_j})} \tag{2.21}$$

By applying mathematical induction based on definition 2.45 and 2.46 it can be proven that the above statements are correct. For details, see lemma A.4.

Definition 2.49 (Performance characteristics of simultaneous transfer *)
Based on lemma A.4, let $\tau_q, \epsilon_q : \mathcal{D}_{st} \to \mathbb{R}_0^+$ functions mapping respectively the transfer time and latency to a simultaneous transfer as:

$$\begin{aligned}\tau_q((h,\delta),q,s,l,r) := &\sum_{j=1}^{m} \chi((t_{j-1},t_j) \notin h)\frac{rKs}{\min(r,\delta_{t_{j-1}},\delta_{t_j})} + \\ &+ \left(\frac{l}{s}-1\right)\max_{j=1}^{m} \chi((t_{j-1},t_j) \notin h)\frac{rKs}{\min(r,\delta_{t_{j-1}},\delta_{t_j})},\end{aligned} \tag{2.22}$$

$$\epsilon_q((h,\delta),q,s,l,r) := \sum_{j=1}^{m-1} \chi((t_{j-1},t_j) \notin h)\frac{rKs}{\min(r,\delta_{t_{j-1}},\delta_{t_j})} \tag{2.23}$$

According to definition 2.42: $\forall i \in [1..r] : \lambda(d_i) = l_d \wedge \lambda(b_i) = l_b$. Therefore, DRC flow can be represented as a simultaneous transfer, since the conditions defined in definition 2.42 ensure that $((h,\delta),q_d,s_d,l_d,r) \in \mathcal{D}_{st}$. Similarly, bulk data flow also can be represented as: $((h,\delta),q_b,s_b,l_b,r) \in \mathcal{D}_{st}$. Having these, the performance functions of a scenario can be defined as follows.

Definition 2.50 (Performance function of DASG DRC transfer)
Let $a := ((h, \delta), \xi, q_d, q_b, w_d, s_d, s_b, l_d, l_b, r) \in \mathcal{AS}$, and $\Gamma_d : \mathcal{AS} \rightarrow \mathbb{R}_0^+$ be a function for determining DRC transfer execution time as:

$$\Gamma_d(a) := \tau_q((h, \delta), q_d, s_d, l_d, r). \tag{2.24}$$

Note that definition 2.49 is applied to identify $\Gamma_d(a)$.

Definition 2.51 (Performance functions of DASG bulk data transfer)
Let $a := ((h, \delta), \xi, q_d, q_b, w_d, s_d, s_b, l_d, l_b, r) \in \mathcal{AS}$, and $\Gamma_b, \Delta_b, \Theta_b : \mathcal{AS} \rightarrow \mathbb{R}_0^+$ be functions for determining respectively bulk data transfer time, overhead, and latency as:

$$\Gamma_b(a) := \tau_q((h, \delta), q_b, s_b, l_b, r), \tag{2.25}$$

$$\Delta_b(a) := \Gamma_b(a) - k_b s_b K, \text{ and} \tag{2.26}$$

$$\Theta_b(a) := \epsilon_q((h, \delta), q_b, s_b, l_b, r). \tag{2.27}$$

Note that definition 2.49 is applied to determine $\Gamma_b(a)$ and $\Theta_b(a)$. $\forall i \in [1..r]$: transferring b_i directly between D_i and A_i takes $\tau_e(\lambda(b_i), (D_i, A_i)) = k_b s_b K$ time. Overhead on bulk data transfer of a particular scenario is considered as this time subtracted from bulk data transfer time.

Definition 2.52 (Overall DASG performance functions)
Let $\Gamma, \Delta, \Theta : \mathcal{AS} \rightarrow \mathbb{R}_0^+$ be functions for determining respectively overall execution time, overhead, and latency of a scenario, where:

$$\Gamma(a) := \Gamma_d(a) + w_d + \Gamma_b(a), \tag{2.28}$$

$$\Delta(a) := \Gamma_d(a) + w_d + \Delta_b(a), \text{ and} \tag{2.29}$$

$$\Theta(a) := \Gamma_d(a) + w_d + \Theta_b(a). \tag{2.30}$$

Note that because DRC transfer always has to be performed before bulk data transfer, DRC transfer time is always added to the overall transfer time, overhead and latency of a scenario.

Definition 2.53 (Scalability of DASG data transfer)
Performance functions of any DASG scenario are characterised based on growth rates in function of l_b and r. This is represented by the Bachmann–Landau (Big O) notation and can have the following values: $O(0)$ if the given performance function is 0; $O(1)$ if it is a positive constant (independent of l_b and r); $O(l_b)$ if it is linear with l_b, but independent of r; $O(r)$ if it is linear with r, but independent of l_b; and finally, $O(rl_b)$ if it is linear with both r and l_b.

2.3.3 DASG bulk data flow

Performance characteristics of bulk data transfer are determined by bulk data path, bulk data staging, instance and type layout. In particular, bulk data path determines through which nodes bulk data is transferred, while instance layout determines whether these nodes are coupled. Obviously, if two nodes are coupled, then no physical data transfer is performed between machines. Based on bulk data path and instance layout, different bulk data flow cases can be identified.

Definition 2.54 (Data flow case *)
A *data flow case* is a (q, π) tuple where $q = (t_0, t_1, ..., t_m) \in \mathcal{L}_P(\mathcal{T})$ is a path layout and $\pi \in [\{(t_{j-1}, t_j) \| j \in [1..m]\} \rightarrow \mathbb{L}]$ is a logical function that defines $\forall j \in [1..m]$: whether $(t_{j-1}, t_j) \in h$. Note that this logical function always defines a set of which elements are instance layouts that satisfy π. A data flow case $((t_0, t_1, ..., t_m), \pi)$ is *representative* if $\forall j \in [1..m] : \pi(t_{j-1}, t_j)$ is false.

According to this definition, 2^m different data flow cases can be defined for path layout $(t_0, t_1, ..., t_m)$ and exactly one of these is representative. In the case of DASGs, there are 2 bulk data path layouts: (D, C, A) and (D, C, M, A). 4 data flow cases can be defined based on (D, C, A) and 8 data flow cases can be defined based (D, C, M, A). However, since DASG instance layouts having (D, C) coupled are not considered, $\pi(D, C)$ should always be false. This means that based on path layout (D, C, A) only 2 data flow cases while based on path layout (D, C, M, A) only 4 data flow cases are considered. These are listed in table 2.3, where representative cases are marked with asterisks. Based on definition 2.15: $\forall(t_1, t_2) \in h : t_1, t_2 \in \mathcal{T} \Rightarrow \delta_{t_1} = \delta_{t_2}$. This means that instance layout implies restrictions on type layout. This is also included in the table.

Definition 2.55 (Data flow case equivalence *)
Data flow case $((t_0, ..., t_{k-1}, t_k, t_{k+1}, ..., t_m), \pi_1)$ and $((t_0, ..., t_{k-1}, t_{k+1}, ..., t_m), \pi_2)$ are *directly equivalent* $(k \in [1..m-1])$ if either $\pi_1(t_{k-1}, t_k)$ or $\pi_1(t_k, t_{k+1})$ is true, $\forall j \in [1..k-1] \cup [k+2..m] : \pi_1(t_{j-1}, t_j) = \pi_2(t_{j-1}, t_j)$ and $\pi_2(t_{k-1}, t_{k+1}) = \pi_1(t_{k-1}, t_k) \wedge \pi_1(t_k, t_{k+1})$. Two data flow cases are *equivalent* if they are in the same equivalence class based on the transitive, reflexive, symmetric closure of this relation.

Based on their equivalence, DASG bulk data flow cases can be divided into 3 different groups, data flow cases in each group are equivalent. These groups are also illustrated in the table 2.3.

Lemma 2.4 (Simultaneous transfer of equivalent data flow cases *)
Let (q_1, π_1) and (q_2, π_2) be equivalent data flow cases and let $((h_1, \delta), q_1, s, l, r)$, $((h_2, \delta), q_2, s, l, r) \in \mathcal{D}_{st}$. If h_1 and h_2 are elements of the instance layout sets that π_1 and π_2 define (respectively), then performance functions of the two simultaneous transfers are the same.

Case	Bulk data path	Instance layout	Restrictions	Group
BC1 *	(D, C, A)	$(C, A) \in h$	$\delta_C = r$	BG1
BC2	(D, C, M, A)	$(C, M), (M, A) \in h$	$\delta_C, \delta_M = r$	
BC3 *	(D, C, A)	$(C, A) \notin h$		BG2
BC4	(D, C, M, A)	$(C, M) \notin h \wedge (M, A) \in h$	$\delta_M = r$	
BC5	(D, C, M, A)	$(C, M) \in h \wedge (M, A) \notin h$	$\delta_M = \delta_C$	
BC6 *	(D, C, M, A)	$(C, M), (M, A) \notin h$		BG3

Table 2.3: DASG bulk data flow cases

Proof is provided in lemma A.5. In the case of DASGs, there are 3 groups. Performance properties of data flow cases are the same within each group. Bulk data transfer time, overhead, and latency values can be found in table 2.4, 2.5, and 2.6 respectively, where the formulas are given by $\Gamma_b(a)$, $\Delta_b(a)$, and $\Theta_b(a)$. Based on $\Delta_b(a)$ and $\Theta_b(a)$, the architectural conditions which determine scalability in terms of latency and overhead are identified for each group, these can be found in table 2.5, and 2.6.

Group	Transfer time ($\Gamma_b(a)$)
BG1	$k_b s_b K$
BG2	$\frac{r(k_b+1)s_b K}{\min\{r,\delta_C\}}$
BG3	$\frac{r s_b K}{\min\{r,\delta_M\}} + \frac{r s_b K}{\min\{r,\delta_C\}} + \frac{r k_b s_b K}{\min\{r,\delta_M,\delta_C\}}$

Table 2.4: Time of DASG bulk data transfer

Group	Overhead ($\Delta_b(a)$)	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
BG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG2	$\frac{r(k_b+1)s_b K}{\min\{r,\delta_C\}} - k_b s_b K$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$	$k_b = 1$ $\delta_C \geqq r$	$\nexists$	$\delta_C < r$
BG3	$\frac{r s_b K}{\min\{r,\delta_M\}} + \frac{r s_b K}{\min\{r,\delta_C\}} + \frac{r k_b s_b K}{\min\{r,\delta_M,\delta_C\}} - k_b s_b K$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$k_b = 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$\nexists$	$\delta_C < r \vee \delta_M < r$

Table 2.5: Overhead and scalability of DASG bulk data staging

In particular, cases of group BG1 always provide 0 overhead and latency on bulk data staging, while cases of group BG2 and BG3 never. In the cases of the latter

Group	Latency ($\Theta_b(a)$)	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
BG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG2	$\frac{rs_bK}{\min\{r,\delta_C\}}$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$	$k_b = 1$ $\delta_C \geqq r$	$k_b > 1$ $\delta_C < r$	$k_b = 1$ $\delta_C < r$
BG3	$\frac{rs_bK}{\min\{r,\delta_M\}} + + \frac{rs_bK}{\min\{r,\delta_C\}}$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$k_b = 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$k_b > 1$ $\delta_C < r \vee \delta_M < r$	$k_b = 1$ $\delta_C < r \vee \delta_M < r$

Table 2.6: Latency and scalability of DASG bulk data staging

two groups the following rules apply: As long as r is not greater than any of the δ values in the overhead formula of a particular group, then overhead is independent of r. If this is the case, then pipelined transfer ($k_b > 1$) implies that overhead is independent of l_b as well, while non pipelined transfer ($k_b = 1$) implies that overhead is linear with l_b. Scalability in terms of overhead is $O(1)$ in the case of the former and $O(l_b)$ in the case of the latter. If r is greater than any of the δ values in the overhead formula of a particular group, then overhead is linear with both r and l_b. In this case scalability in terms of overhead is $O(rl_b)$. Note that none of the architectures realise scalability $O(r)$ in terms of overhead.

Similar rules apply for scalability in terms of latency with the following differences. If r is greater than any of the δ values in the latency formula of a particular group, then latency is linear with r. If this is the case, then pipelined transfer ($k_b > 1$) implies that latency is independent of l_b, while non pipelined transfer ($k_b = 1$) implies that latency is linear with l_b. Scalability in terms of latency is $O(r)$ in the case of the former and $O(rl_b)$ in the case of the latter.

2.3.4 DASG DRC flow

Performance characteristics of DRC transfer are determined by DRC path, DRC staging, instance and type layout. DRC flow analysis is similar to bulk data flow analysis in several aspects. Based on DRC path layout (R, C) and (R, M, C) 6

different cases can be identified. These are listed in table 2.7. Instance layout implies restrictions on type layout in case DC1, DC2, DC4, and DC5.

Case	DRC path	Instance layout	Restrictions	Group
DC1 *	(R, C)	$(R, C) \in h$	$\delta_R = \delta_C$	DG1
DC2	(R, M, C)	$(R, M), (M, C) \in h$	$\delta_C = \delta_M = \delta_R$	
DC3 *	(R, C)	$(R, C) \notin h$		DG2
DC4	(R, M, C)	$(R, M) \in h \wedge (M, C) \notin h$	$\delta_M = \delta_R$	
DC5	(R, M, C)	$(R, M) \notin h \wedge (M, C) \in h$	$\delta_C = \delta_M$	
DC6 *	(R, M, C)	$(R, M), (M, C) \notin h$		DG3

Table 2.7: DASG DRC data flow cases

Based on data flow case equivalence (see definition 2.54), the 6 DRC data flow cases can be divided into 3 groups. According to lemma A.5, performance properties are the same within each group. Transfer time values are determined based on the definition of $\Gamma_d(a)$ and included in table 2.8. Scalability is only analysed in function of r, since l_b does not affect DRC flow.

Group	Transfer time ($\Gamma_d(a)$)	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
DG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
DG2	$\frac{rk_d s_d K}{\min\{r, \delta_C, \delta_R\}}$	$\nexists$	$\delta_R \geqq r$ $\delta_C \geqq r$	$\nexists$	$\delta_R < r \vee \delta_C < r$	$\nexists$
DG3	$\frac{rs_d K}{\min\{r, \delta_R, \delta_M\}} +$ $+ \frac{rs_d K}{\min\{r, \delta_M, \delta_C\}} +$ $+ \frac{r(k_d - 1)s_d K}{\min\{r, \delta_M, \delta_R, \delta_C\}}$	$\nexists$	$\delta_M \geqq r$ $\delta_C \geqq r$ $\delta_R \geqq r$	$\nexists$	$\delta_M < r \vee \delta_R < r \vee \delta_C < r$	$\nexists$

Table 2.8: Time and scalability of DASG DRC transfer

Since, for all data flow cases in DG1 transfer time of DRC transfer is 0, scalability is $O(0)$. In cases of group DG2 and DG3 the following rules apply. As long as r is not greater than any of the δ values in the transfer time formula of a particular group, than transfer time is independent of r. In this case scalability is $O(1)$. If r is greater than any of the δ values in the transfer time formula of a particular group, than overhead is linear with r. In this case scalability is $O(r)$. Scalability values of DRC transfer are listed in table 2.8.

2.3.5 Recommended DASG structure layout, data flow, and resource layout

In any of the 6 bulk data transfer cases, it is possible to realise scalable bulk data transfer where overhead and latency on bulk data transfer are both independent of r and l_b. However, not under the same conditions. In terms of type layout, while cases BC1 and BC3 require that $\delta_C \geqq r$, cases BC2, BC4, BC5, and BC6 also require that $\delta_M \geqq r$. In terms of bulk data staging BC3, BC4, BC5, and BC6 require pipelined staging ($k_b > 1$) as well. In all of the 6 DRC transfer cases DRC transfer time is independent of l_b and in any of the 6 cases it is possible to realise scalable DRC transfer, where DRC transfer time is independent of r. DC1 and DC2 always provide scalable DRC transfer. In terms of type layout, DC3, DC4 provides scalable DRC transfer if $\delta_C \geqq r$ and $\delta_R \geqq r$, while DC5 and DC6 also require that $\delta_M \geqq r$. The selection of proposed structure layout, resource layout, and data flow combinations is based on the following recommendations:

R1 - Mediator Based section 2.3.1, the mediator is recommended to be hosted as a centralised service on a dedicated machine ($\xi_M = DeM$). Based on this and on definition 2.32, the larger is the number of utilised dedicated machines, the more cost demanding it is to set up and maintain a DASG. For this reason, it is aimed to minimise δ_M (let $\delta_M = 1$). Therefore, data flow cases that require that $\delta_M \geqq r$ are not recommended. On the one hand, in each scenario $\delta_A = r$ is always true. Hence, data flow cases which require to have (A, M) coupled are not recommended. On the other hand, since bulk data amount is multiple orders of magnitude greater than DRC data amount, it is always aimed to realise bulk data transfer of which overhead and latency is independent of the number of simultaneous requests and bulk data amount. This means that data flow cases which are based on bulk data path layout (D, C, M, A) are not recommended and $\delta_C \geqq r$ should always be provided which

implies that data flow cases which require to have $(M, C) \in h$ are not recommended either.

R2 - DRC repository Instance layouts having (A, R) coupled are not recommended, since it cannot be guaranteed that the machine that hosts the application has a copy of the required DRC to access a particular data resource. Note, that this also implies that $\xi_R \neq ExM$. Instance layouts having (M, C) coupled are not recommended based on R1. In the case of instance layouts requiring both $(A, C) \in h$ and $(R, C) \in h$ are not recommended because $(A, R) \in h$ is not recommended and h is transitive. If $(A, C) \notin h$, then (R, C) can be realised only by utilising dedicated machines for running nodes of R and C. This is not advised since the number of dedicated machines should be minimised, but in order to provide a scalable solution $\delta_C \geqq r$ should be provided based on recommendation R1. Therefore, $(R, C) \in h$ is not recommended in general.

R3 - DRC execution Instance layouts requiring $(R, C) \in h$ are not recommended based on R2 and instance layouts requiring $(M, C) \in h$ are not recommended based on R1. $(A, C) \in h$ implies that the DRC has to be transferred to the application machine ($\xi_C = ExM$), while $(A, C) \notin h$ implies that the DRC is either transferred to a dedicated machine ($\xi_C = DeM$) or to a computational machine for execution ($\xi_C = CoM$). $\xi_C = DeM$ is not recommended, since based on R1 the number of utilised dedicated machines should be minimised, but $\delta_C \geqq r$ is aimed to be provided. In the case of a typical Grid environment, using computational machines for DRC execution ($\xi_C = CoM$) adds delay on overhead and latency resulted by the fact that the DRC executable has to wait in a job queue before it is scheduled for execution. (This delay is represented by $w_d \geqq 0$.) However, if $\xi_C = ExM$, then since the application is already running, it can execute the DRC as soon as it is received without being held in a job queue. There is no delay that increases overhead and

Recommendation	DASG data flow case
R1	BC2, BC4, BC5, BC6, DC2, DC5
R2	DC1, DC2
R3	BC3
R4	DC6

Table 2.9: Elimination of DASG data flow cases based on different recommendations.

latency in this case ($w_d = 0$) and $\delta_C = r$ is always provided. Therefore, only instance layouts that have (A, C) coupled are recommended.

R4 - DRC transfer In the case of instance layouts requiring that $(R, M) \notin h$ DRCs should not be transferred via the mediator, because this would increase overhead and if $\delta_R > 1$ the mediator would also bottleneck data transfer in case of multiple simultaneous requests. Therefore, if $(R, M) \notin h$, then only DRC path type (R, C) is recommended.

Having these, data flow cases listed in the right column of table 2.9 are excluded based on the recommendations indicated in the left column. This means, that only bulk data flow case BC1 combined with DRC flow case DC3 or DC4 can be recommended. If it is not possible to utilise multiple Grid storage machines for hosting the DRC repositories then it is recommended to host the DRCs at the mediator machine ($(M, R) \in h, \delta_R = 1, \xi_R = DeM$), since the mediator will not bottleneck DRC transfer. In this case both DC3 and DC4 can be recommended. However, if it is possible to utilise multiple Grid storage machines for hosting DRC repositories then it is recommended to have a decoupled mediator ($(M, R) \notin h, \delta_R \geqq 1, \xi_R = StM$). In this case only DC3 can be recommended.

Based on these the recommended architectures realise either of the structure layout, data flow, and resource layout combinations represented in table 2.10. Combinations of these structure layouts and representative data flow cases are illustrated in figure 2.5. Whether DCR path in the case of PC1 is (R, C) or (R, M, C) is irrel-

Aspects			Proposed 1 (PC1)	Proposed 2 (PC2)
Structure	Instance layout	(A,M)		
		(A,C)	X	X
		(A,R)		
		(M,R)	X	
		(R,C)		
		(C,M)		
	Type layout	δ_M	1	1
		δ_R	1	≥1
		δ_C	r	r
Data flow	Bulk data path	(D,C,A)	X	X
		(D,C,M,A)		
	Bulk data staging	Pipelined (k_b=1)	O	O
		Not pipelined (k_b>1)	O	O
	DRC path	(R,C)	O	X
		(R,M,C)	O	
	DRC staging	Pipelined (k_d>1)	O	O
		Not pipelined (k_d=1)	O	O
Resource	Resource layout	Mediator	DeM	DeM
		DRC Repository	DeM	StM
		DRC execution	ExM	ExM
Performance	Overhead		rl_d	$\frac{rl_d}{\min(r,\delta_R)}$
	Overhead scalability		O(r)	O(r)
	Latency		rl_d	$\frac{rl_d}{\min(r,\delta_R)}$
	Latency scalability		O(r)	O(r)

Table 2.10: Recommended DASG structure, data flow, and resource layout combinations, where sign X shows that a particular architectural approach is realised by the correlated combination. Multiple X signs within instance layout indicate multiple couplings at the same time. Multiple O signs within data flow indicate that there is no difference between the correlated approaches of a particular combination.

evant in terms of performance, since DC3 and DC4 are equivalent based on definition 2.55. Bulk data staging and DRC staging are also irrelevant, since in the case of BC1, DC3, and DC4, there is no difference in performance between pipelined and non pipelined transfer according to lemma A.3.

The key difference between the two proposed combinations is that while PC1 uses only one repository that is coupled with the mediator, PC2 can use multiple decoupled repositories. This results in different performance characteristics. In the case of PC1 overall overhead and latency both equal to rl_dK, while in the case of PC2 overall overhead and latency both equal to $\min\{1, \frac{r}{\delta_R}\}l_dK$. Hence, if multiple storage resources can be provided, then PC2, otherwise PC1 is recommended. Proposed architectures are described and compared to the existing solutions in the next section. (See table 2.11 and figure 2.5.)

2.4 Existing and proposed DASG solutions

2.4.1 Existing DASG solutions

Although there are several solutions for accessing data resources, many of these provide access to specific types of data resources. SAGA, SRB, LFC, Hadoop MapReduce [97] provide access to file system based distributed data resources and focus on additional functionality such as replication and high throughput data retrieval. Although, these solution are aiming to support distributed applications that simultaneously process large data amounts, they cannot be considered as DASGs, since they are not designed to provide access to a wide range of data resources: SQL and XML databases, file systems and repositories.

On the other hand, there are a few existing DASG solutions, such as OGSA-DAI, GRelC, or the AMGA metadata catalogue, as it was mentioned in the introduction of this chapter. OGSA-DAI was already introduced in section 1.4.1.

GRelC (Grid Relational Catalogue) was developed at the CACT/ISUFI Laboratory of the University of Salento, Lecce and the SPACI Consortium. It is compatible with gLite and Globus middleware and provides access to relational and non-relational data resources. It is used by several projects, according to [98]. From numerous aspects it is similar to OGSA-DAI, since it also implements the DAIS specification.

The AMGA metadata catalogue was designed to provide access to metadata for files stored on the Grid, but it also provides simplified access to relational databases of different vendors. Therefore, it is considered as a DASG. It has back-ends for Oracle, PostgreSQL, MySQL, and SQLite. It was developed in collaboration with

Aspects			Existing						Proposed	
		Architectures	OGSA-DAI (not pipelined)	OGSA-DAI (pipelined)	GRelC (not pipelined)	GRelC (pipelined)	AMGA (not pipelined)	AMGA (pipelined)	Proposed 1 (PC1)	Proposed 2 (PC2)
Structure	Instance layout	(A,M)								
		(A,C)							X	X
		(A,R)								
		(M,R)	X	X	X	X	X	X	X	
		(R,C)	X	X	X	X	X	X		
		(C,M)	X	X	X	X	X	X		
	Type layout	δ_M	1	1	1	1	1	1	1	1
		δ_R	1	1	1	1	1	1	1	≥1
		δ_C	1	1	1	1	1	1	r	r
Data flow	Bulk data path	(D,C,A)							X	X
		(D,C,M,A)	X	X	X	X	X	X		
	Bulk data staging	Pipelined (k_b=1)		X		X		X	O	O
		Not pipelined (k_b>1)	X		X		X		O	O
	DRC path	(R,C)	X	X	X	X	X	X	O	X
		(R,M,C)							O	
	DRC staging	Pipelined (k_d>1)	O	O	O	O	O	O	O	O
		Not pipelined (k_d=1)	O	O	O	O	O	O	O	O
Resource	Resource layout	Mediator	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM
		DRC Repository	DeM	DeM	DeM	DeM	DeM	DeM	DeM	StM
		DRC execution	DeM	DeM	DeM	DeM	DeM	DeM	ExM	ExM
Interface	Backend interface	API	X	X	X	X	X	X		
		CLI							X	X
	Frontend interface	General	X	X	X	X			X	X
		Specific					X(SQL)	X(SQL)		
Performance	Overhead		$2rl_b$-l_b	rs_b+rl_b-l_b	$2rl_b$-l_b	rs_b+rl_b-l_b	$2rl_b$-l_b	$2rl_b$-l_b	rl_d	$\frac{rl_d}{\min(r,\delta_R)}$
	Overhead scalability		$O(rl_b)$	$O(rl_b)$	$O(rl_b)$	$O(rl_b)$	$O(rl_b)$	$O(rl_b)$	O(r)	O(r)
	Latency		rl_b	rs_b	rl_b	rs_b	rl_b	rs_b	rl_d	$\frac{rl_d}{\min(r,\delta_R)}$
	Latency scalability		$O(rl_b)$	O(r)	$O(rl_b)$	O(r)	$O(rl_b)$	$O(rl_b)$	O(r)	O(r)

Table 2.11: Analysis of proposed and existing DASG architectures, where signs represent the same concepts as in the case of table 2.10.

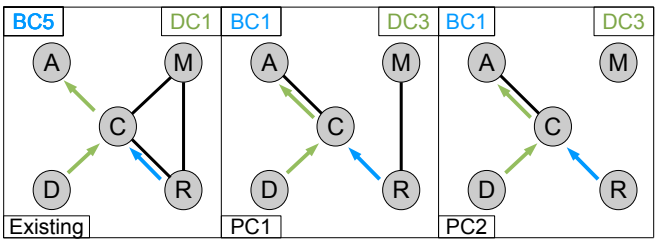

Figure 2.5: Structure and representative data flow cases of the existing and proposed DASG architectures, where black lines represent which node types are coupled, green arrows represent bulk data path layouts and blue arrows represent DRC path layouts.

the EGEE user community and it is the metadata catalogue for gLite. AMGA is planning to adapt the WS-DAIR Standard that is included in the DAIS specification.

Although, these solutions were designed for various purposes, they are very similar. The aim of this research is not to analyse these solutions based on each possible aspect, but to compare them based on generality, extendibility, and performance by identifying what architecture they realise based on the architectural aspects described in section 2.2.

From a structural point of view, OGSA-DAI, GRelC, and AMGA are identical. All solutions integrate and tightly couple the mediator (M), the DRC repository (R), and the DRC execution (C). Hence instance layout of all existing solutions is $h = \overline{\{(M,R)}, \overline{(R,C),(C,M)\}}$, implying that, in terms of type layout: $\delta_M = \delta_R = \delta_C$. By default, existing solutions are installed on a single computer, hence, this is number is 1.

In terms of data flow, existing solutions are also identical. Since repository and execution are coupled, DRC is not staged physically from one computer to another. They all implement DRC flow case DC1. This means, that performance formulas of DG1 in table 2.8 apply meaning that there is no overall overhead or latency increase resulted by DRC transfer and also implies that DRC staging is irrelevant, since there is no difference in performance between pipelined and non pipelined transfer according to lemma A.3. DRCs are executed on a dedicated machine, no job queues are involved. Therefore, $w_d = 0$ is always true. Bulk data path of the existing solutions implement BC5. The performance formulas of group BG2 in table 2.4, 2.5, and 2.6 apply. All existing solutions can realise both pipelined and non pipelined bulk data transfer. Pipelined transfer is typically realised by data streaming if it is supported by the accessed data resource.

In the case of all existing solutions, all core nodes are hosted on the same dedicated machine. Therefore, in terms of resource layout: $\xi_M = \xi_R = \xi_C = DeM$.

The driver interface, through which the DRCs are accessed is based on APIs. In most cases it is partially based on existing frameworks such as JDBC (OGSA-DAI) or ODBC (GRelC, AMGA). Application interface is general for most solutions, except AMGA, that is specific to SQL.

Architectures that the exsiting solutions can realise are included in table 2.11 and also illustrated in figure 2.5.

2.4.2 Proposed DASG solutions

Based on the analysis in section 2.3, two types of architectures are proposed. These also can be seen in table 2.11. In the case of the proposed architectures, DRCs are accessed via CLI. This helps the straightforward integration of a new data resource. Note that CLI is the native interface for executing applications on the majority of Grid provided computational machines. Therefore, using CLI also simplifies DRC execution on these machines. On the other hand, frontend interface is general, hence, does not restrict the set of data resources that can be accessed via a DASG. They implement either of the structure, data flow, and resource layout combinations described in table 2.10. The recommended DASG solution should realise one or more of the proposed architectures depending on what cases it will be used in.

2.4.3 Comparison of existing and proposed DASG solutions

The key difference between existing and proposed architectures lies in their structure and resource layout. In the case of all existing solutions mediator, repository, and DRC execution are coupled and hosted by the same dedicated machine. This minimizes DRC transfer time, since DRCs do not have to be transferred between any

machines, but requires bulk data to flow through this machine, resulting in this machine bottlenecking data transfer. Proposed solutions couple only application and DRC execution. Hence, DRC has to be transferred from a DRC repository machine to an application machine. With other words, although both existing and proposed solutions provide a centralised mediator, in the case of the proposed solutions the DRC is moved to the application, removing the centralised mediator from the path of bulk data. This results in increased DRC transfer time, but as soon as the DRC is at the application machine, bulk data flows directly between this machine and the data resource without additional overhead on bulk data transfer.

As a result overall overhead of existing architectures is linear with both number of simultaneous requests (r) and bulk data size (l_b), while overhead of proposed architectures is independent of l_b and linear with r in the case of both PC1 and PC2. However, in the case of PC2, as long as $\delta_R \geqq r$, overall overhead is constant l_d, hence, independent of r. Comparing overhead of existing and proposed architectures, overhead of not pipelined transfer through the existing architectures is $\left(2 - \frac{1}{r}\right) \frac{l_b}{l_d}$, while overhead of pipelined transfer through existing architectures is $\frac{s_b}{l_d} + \left(1 - \frac{1}{r}\right) \frac{l_b}{l_d}$ times more, than overhead of transfer through PC1. Furthermore, overhead of not pipelined transfer through the existing architectures is $\min(\delta_R, r) \left(2 - \frac{1}{r}\right) \frac{l_b}{l_d}$, while overhead of pipelined transfer through existing architectures is $\min(\delta_R, r) \frac{s_b}{l_d} + \min(\delta_R, r) \left(1 - \frac{1}{r}\right) \frac{l_b}{l_d}$ times more, than overhead of transfer through PC1.

Since $\frac{s_b}{l_d}$ is constant and assuming that r is sufficiently large, overhead of not pipelined transfer through existing architectures is about $2\frac{l_b}{l_d}$, while overhead of pipelined transfer through existing architectures is roughly $\frac{l_b}{l_d}$ times greater, than overhead of transfer through PC1. Moreover, overhead of not pipelined transfer through existing architectures is about $2\delta_R \frac{l_b}{l_d}$, while overhead of pipelined transfer through existing architectures is roughly $\delta_R \frac{l_b}{l_d}$ times greater, than overhead of transfer through PC2.

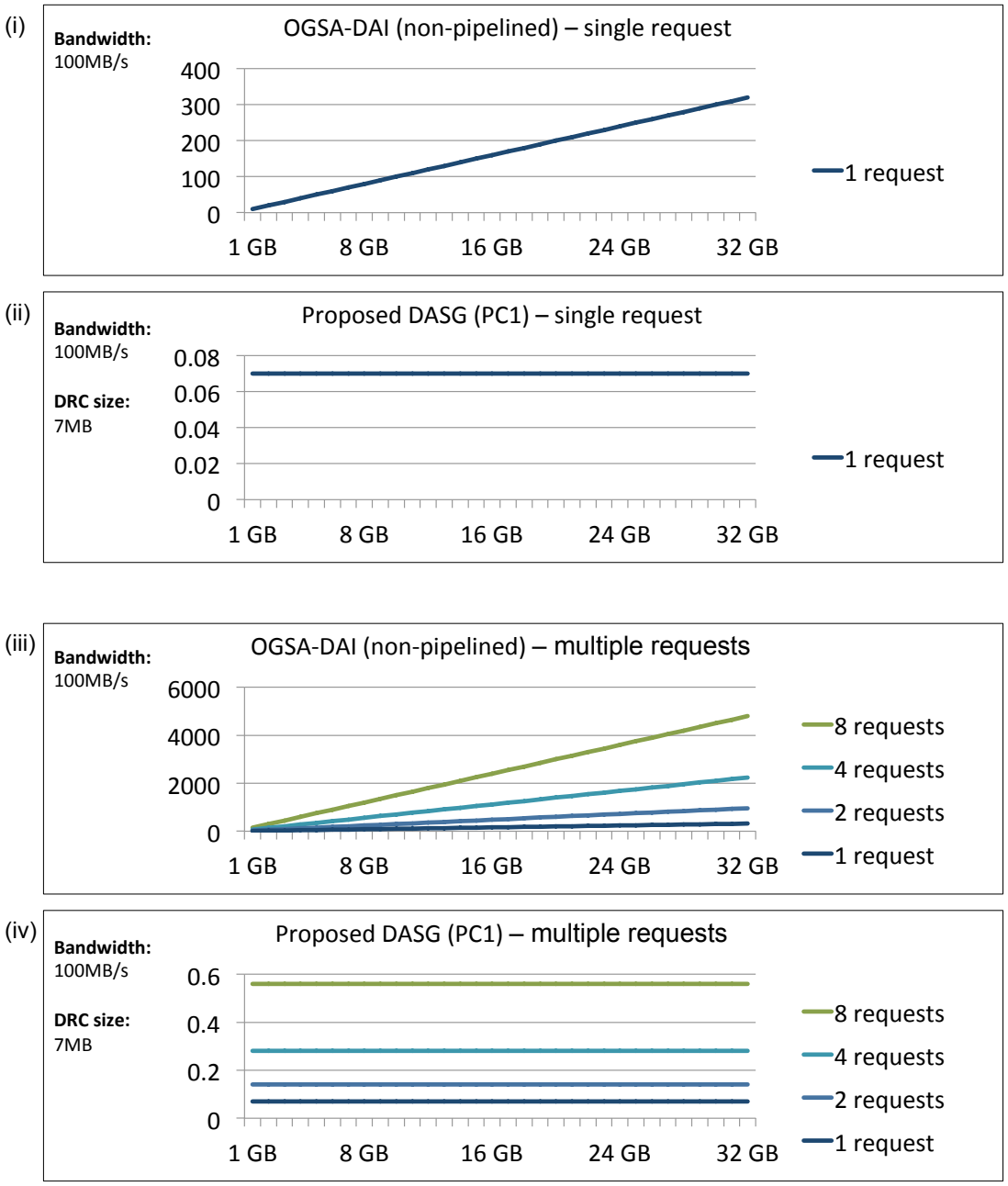

Figure 2.6: Overall overhead predictions of OGSA-DAI (non-pipelined) and Proposed DASG (PC1) architectures in seconds in the function of bulk data size, where graph (i) and (ii) represent single, graph (iii) and (iv) represent multiple request executions.

The difference in size between DRC and bulk data is significant (l_d is measured in MB, l_b is typically measured in GB or TB). Both PC1 and PC2 show significant performance improvement comparing to existing solutions.

Figure 2.6 compares overhead predictions of the OGSA-DAI (non-pipelined) architecture and Proposed DASG (PC1) architecture in the case of a single and multiple requests in function of bulk data amount. The graphs represent the overall overhead performance formulas provided for these architectures in table 2.5. The performance predictions are based on a network with 100MB/s bandwidth and 7MB DRC size. As graph (i) shows, overhead of the OGSA-DAI (non-pipelined) architecture increases linearly with bulk data size. In the case of 32GB, overhead is above 300s (5 minutes). This is resulted by the fact that first bulk data is transferred to the OGSA-DAI service machine and then transferred further to the application machine. In comparison, as graph (ii) shows, overhead of the Proposed DASG architecture (PC1) is 0.07s. This is the amount of time required for the DRC to be transferred from the DRC repository machine to the application machine. Since after this point bulk data is transferred directly between the data resource machine and the application machine, overhead is constant and independent of bulk data size. Graph (iii) and (iv) show that overhead of both architectures increase linearly with request number. However, while in the case of the OGSA-DAI (non-pipelined) architecture with 8 simultaneous requests and 32GB bulk data size overhead is nearly 5000s (about 83 minutes), overhead of the Proposed DASG architecture (PC1) is below 0.7s.

2.5 Implementation

To show that it is possible to realise the distributed concept of the proposed architectures, a solution was developed for accessing heterogeneous data based on

GEMLCA [99, 100]. GEMLCA is unique in a sense that it is an application repository extended with a job submitter. It allows the deployment of legacy code applications on the Grid. An application can be exposed via a GEMLCA service and can be executed using a GEMLCA client. The legacy application is stored either in the repository of a GEMLCA service or on a third party computational node where GEMLCA can access it. To publish a legacy application via GEMLCA, only a basic user-level understanding of the legacy application is needed, code re-engineering is not required. As soon as the application is deployed, GEMLCA is able to submit it using either GT2, GT4 [10] or gLite [12] Grid middleware. GEMLCA also provides a list of computational sites where the legacy application in question can be executed (these sites are defined by the application owner that publishes the legacy application) and allows scientists to select a suitable site. For the same legacy application, GEMLCA allows the specification of different binaries and different configurations for different sites.

A command line MySQL client was deployed in GEMLCA as a legacy application. 6 command line parameters (database hostname, username, password, database name, SQL input file, and result file) of the application were defined in GEMLCA via the GEMLCA administration portlet, that provides a web based graphical user interface and can be used either as a stand alone portlet or it can be integrated to any JSR-168 [101] based portal. See illustration in figure 2.7.

Since GEMLCA was designed to execute legacy applications on remote computational machines, the GEMLCA client was extended with the capability to execute DRCs locally to the GEMLCA client. The GEMLCA based DASG solution is illustrated in figure 2.8. The GEMLCA service that realises the mediator also encapsulates an application repository which is used as the DRC repository. This also means that in this solution the mediator and the DRC repository are coupled. First, the Grid application passes its data request to the GEMLCA client. (This is

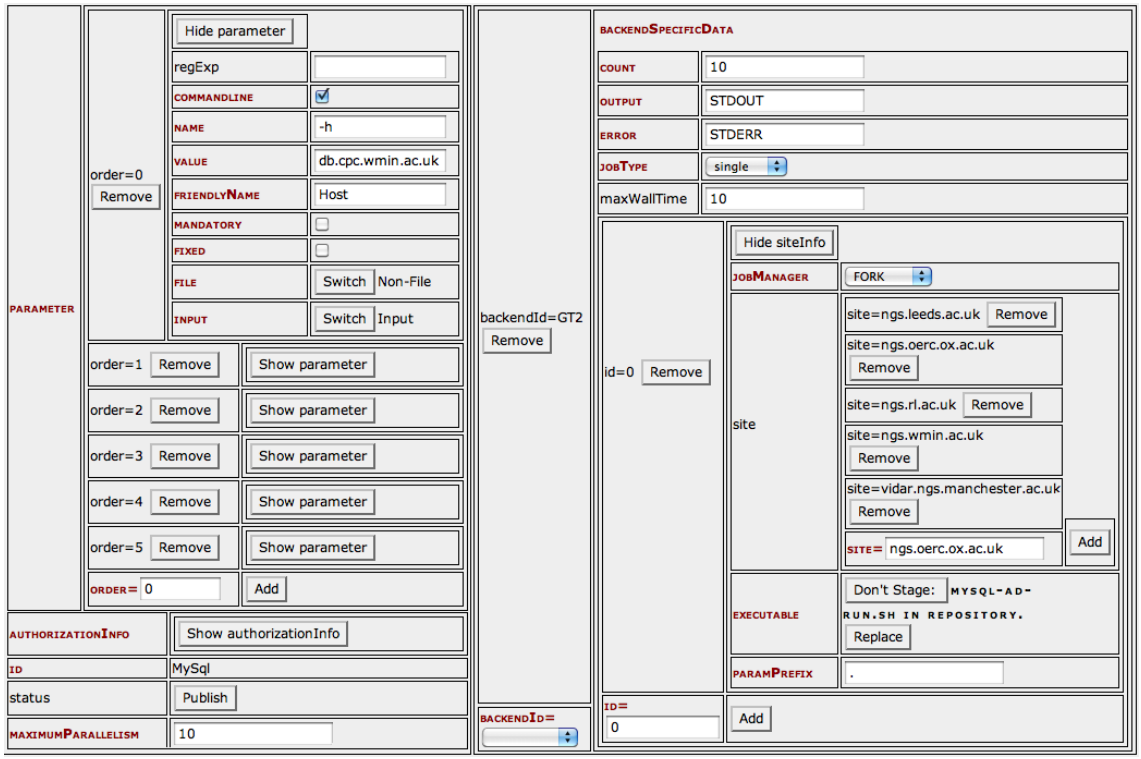

Figure 2.7: Deploying MySQL client using the GEMLCA Administration Portlet

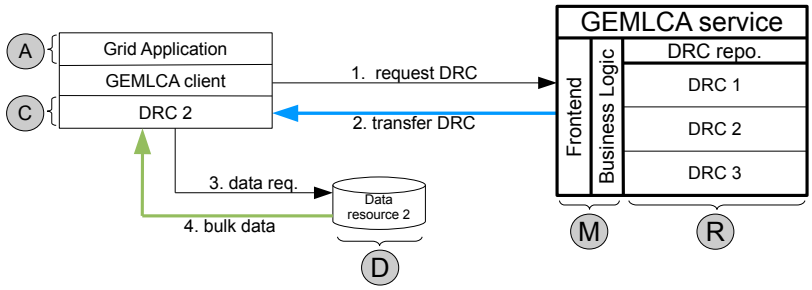

Figure 2.8: Implementation of DASG PC1 based on GEMLCA, where black arrows represent control data, blue arrow represents DRC transfer, and green arrow represents bulk data transfer.

not shown on the figure). In the case of MySQL this request includes the location of the GEMLCA service where the MySQL client is deployed and the 6 parameters of the MySQL client. The GEMLCA client selects the appropriate MySQL client

binary and sends a request to the GEMLCA service to transfer it. When the transfer finishes, the GEMLCA client parametrises and executes the MySQL client, which sends the request to the selected database and receives the result data set. This way bulk data flows directly between the database and the machine that runs the Grid application. This solution implements PC1.

2.6 Summary

This chapter proposes two types DASG architectures that provide access to heterogeneous data resources for Grid applications and introduces a general mathematical model which is also utilised in the following chapters. Although there is a clear demand for a general, easily extendible, and scalable solution that provides access to heterogeneous data resources for Grid applications with low overhead, currently there is no such solution available for Grid users.

The described architecture analysis not only compares proposed architectures and architectures of existing solutions, but also compares numerous other possibilities at the level of data flow. This data flow analysis can be utilised in special scenarios which are not addressed by this research.

Based on the proposed architectures, scalable solutions can be realised, which allow direct data transfer with low overhead. This can be realised by an approach where DRCs are dynamically distributed and executed on the machine where the bulk data is generated/processed by the Grid application. Distribution of DRCs increases overhead, but this is minimal compared to bulk data transfer time due to the small size of the DRCs. Bulk data is transferred directly between this machine and the data resource machine.

In contrast to the proposed architectures, all existing solutions that could be

used as a DASG (OGSA-DAI, AMGA, and GRelC) host and execute DRCs on a dedicated machine, where bulk data is first transferred to this machine, then to the machine that runs the Grid application. This increases overhead and bottlenecks data transfer in the case of large number of requests.

The GEMCLA based reference implementation shows that the proposed architectures are possible to realise. This solution is easily extendible with any DRC that has a command line interface. (This is provided in most cases.) Adding a new DRC to GEMLCA can be done using a simple graphical user interface, only a basic user-level understanding of DRC is needed, code re-engineering is not required. Furthermore, it is a general solution, since it has a generic frontend interface which do not restrict the number and type of input parameters that can be passed to the DRC. The only restriction is that the parameters have to be represented either as command line arguments or files.

Chapter 3

Heterogeneous Data Access Solutions for Workflows (DASW)

Although many scientific experiments rely on data stored in various data resources, the capability of most workflow management systems to access a large set of data resource types during workflow execution is very limited. (See table 1.1.) For this reason, scientists have to use different tools before workflow submission to access their data-sets and retrieve the required data on which they want to carry out computational experiments.

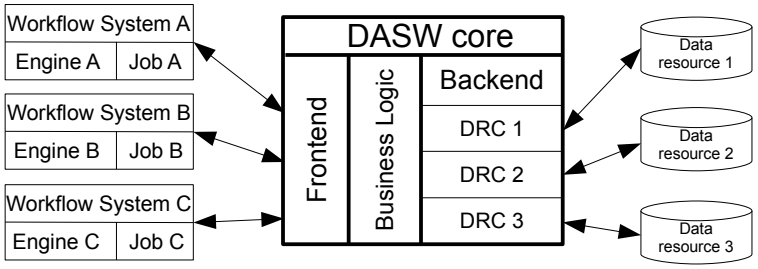

Figure 3.1: Concept of existing DASWs

A DASW has a very similar concept to a DASG with the exception that here data

access is provided not for Grid applications but for workflows of different workflow systems. See illustration in figure 3.1 Although there are no general solutions for workflow systems to access heterogeneous data, some workflow systems support the access to different kinds of data resources. These are detailed and analysed in section 3.4.

3.1 Key DASW properties and requirements

Six key properties of DASWs were identified. These are: (i) generality, (ii) extendibility, (iii) overhead, (iv) latency, (v) scalability and (vi) data access. Properties (i-v) are important in the case of DASWs for the same reasons as in the case of DASGs described in section 2.1.

Data access *Data access* is a DASW property that indicates when a data resource can be accessed by a workflow. Data access can be: (i) static, (ii) semi-dynamic, or (ii) dynamic. See figure 3.2. *Static* approach means that the data resource can be accessed before and after workflow execution, but it cannot be accessed at workflow runtime. For instance, data is gathered from a database before execution, stored in a file that will act as an input for the workflow. Similarly to this procedure, data that is obtained as a result of workflow execution can be transferred to a database. In the case of *semi-dynamic* approach, data resource is accessed during workflow execution. However, the parameters of the data request are already specified before execution and cannot be generated at runtime. Although all the request parameters are determined when the workflow execution starts, data itself will be transferred at execution time, ensuring that the workflow nodes will receive the most recent data

content before they start their computation and the data will appear instantly in the data resource when the node finishes its computation. *Dynamic* approach enables access to data resource at workflow runtime and the parameters of the request are also generated during workflow execution. This approach gives the greatest flexibility, since not only the data content is transferred in a dynamic fashion, but the data request itself can be generated by the same workflow. Therefore, it is aimed to find a solution that supports dynamic data access for workflows.

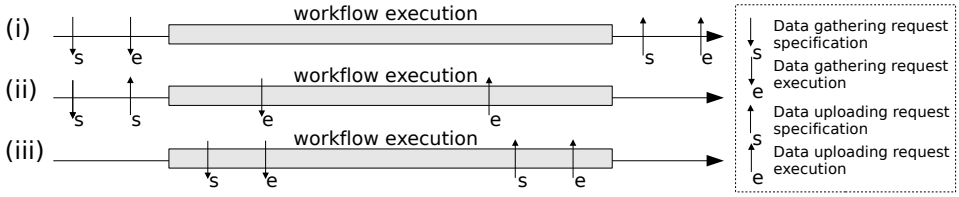

Figure 3.2: DASW Data access approaches: (i) static, (ii) semi-dynamic, and (iii) dynamic

3.2 DASW architecture definition

This section defines how DASW architectures are represented based on the general definitions (marked with asterisks) introduced in chapter 2. DASW architectures are defined based on five properties: *structure*, *data flow*, *resources*, *interface*, and *integration*. These are defined in the followings.

Similarly to DASGs, existing DASWs encapsulate their complete functionality on a single machine. How the load on this machine can be distributed by utilising computational and storage resources provided by the Grid is under investigation in the followings.

3.2.1 DASW node

Definition 3.1 (DASW nodes and node types)
In the case of DASWs, six node types are distinguished. *Mediator*, *DRC repository*, *DRC execution*, and *data resource* nodes and node types represent the same concepts as defined in the case of DASGs in definition 2.2. In addition:

- An *engine node* represents a running workflow engine. Engine nodes belong to node type W.
- A *job node* represents a running workflow job that has either generated data or it is going to process some data that is to be transferred to/from a data resource. Job nodes belong to node type J.

Let $\mathcal{T} := \{W, J, M, R, C, D\}$ be the *set of all DASW node types*. Nodes of M, R, C provide a DASW service and enable the communication between the nodes of J and D. Therefore, two disjunctive subsets can be identified within $\mathcal{T}$: let $\mathcal{T}' := \{M, R, C\}$ be the *set of core DASW node types* and let $\mathcal{T}'' := \{W, J, D\}$ be the *set of external DASW node types*. See illustration on figure 3.3.

Note that a workflow engine is the software component that orchestrates the execution of workflow jobs and is responsible for the execution of the whole workflow. The term "workflow job" refers to a job which is part of a given workflow.

Definition 3.2 (DASW instance)
Let a *DASW instance* be a set of $|\mathcal{T}| = 6$ nodes, where each node belongs to a different node type of $\{W, J, M, R, C, D\}$.

Definition 3.3 (Bijection between DASW node types and instances)
Let $\forall i \in [1..r]$: let $\mathcal{N}_i := \{W_i, J_i, M_i, R_i, C_i, D_i\}$ be the ith DASW instance, where $\varphi_i(W) = W_i$, $\varphi_i(J) = J_i$, $\varphi_i(M) = M_i$, $\varphi_i(R) = R_i$, $\varphi_i(C) = C_i$, and $\varphi_i(D) = D_i$.

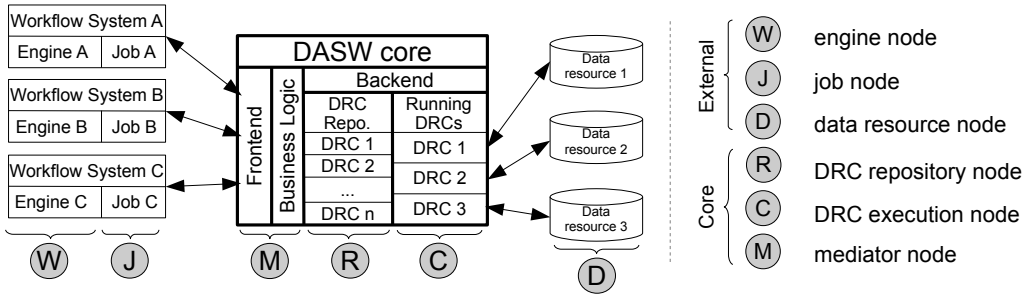

Figure 3.3: DASW node types

In the case of DASWs there are 6 node type sets $\mathcal{N}_W$, $\mathcal{N}_J$, $\mathcal{N}_M$, $\mathcal{N}_R$, $\mathcal{N}_C$, $\mathcal{N}_D$ and r nodes in each type set. A DASW node matrix of instances and types can be constructed as illustrated in table 3.1. Furthermore, both $\bigcup_{i=1}^{r} \mathcal{N}_i$ and $\bigcup_{t \in \mathcal{T}} \mathcal{N}_t$ are equal to the set of all DASW nodes, $\mathcal{N}$ and $|\mathcal{N}| = 6r$.

		Node types					
		$\mathcal{N}_W$	$\mathcal{N}_J$	$\mathcal{N}_M$	$\mathcal{N}_R$	$\mathcal{N}_C$	$\mathcal{N}_D$
Instances	$\mathcal{N}_1$	W_1	J_1	M_1	R_1	C_1	D_1
	$\mathcal{N}_2$	W_2	J_2	M_2	R_2	C_2	D_2
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
	$\mathcal{N}_r$	W_r	J_r	M_r	R_r	C_r	D_r

Table 3.1: DASW node matrix

3.2.2 DASW structure

Similarly to DASGs, DASW structure layouts are defined by instance and type layout.

Definition 3.4 (DASW instance layout)
The set of all possible DASW instance layouts is represented by $\mathcal{L}_I$ and equals to the set of all possible type layouts on domain $\mathcal{T}$ (see definition 2.11).

Definition 3.5 (DASW type layout)
The set of all possible DASW type layouts is represented by $\mathcal{L}_T$ and equals to the set of all possible type layouts on domain $\mathcal{T}$ (see definition 2.13).

Definition 3.6 (DASW structure layout)
The set of all possible DASW structure layouts is represented by $\mathcal{L}_S$ and equals to the set of all possible structure layouts on domain $\mathcal{T}$ (see definition 2.15).

3.2.3 DASW data flow

Definition 3.7 (DASW Data types)
Similarly to the case of DASGs, in the case of DASWs three kinds of data are distinguished:

- *bulk data* is the data-set that needs to be transferred between (D_i) and (J_i);
- *DRC data* is the DRC itself that needs to be transferred from (R_i) to (C_i); and
- *control data* is the set of information that includes all further data transferred between the nodes. The latter consists of a small number of requests which are necessary to exchange in order to provide access for an application to a data resource.

Two kinds of data flow are considered: *DRC flow*, and *bulk data flow*. *Control flow* is excluded from the model for the same reasons as in the case of DASGs, described in section 2.2.3.

Definition 3.8 (DASW DRC flow)

Four DRC path layouts are distinguished in the case of DASWs: when the DRC is transferred directly (R, C), when it is transferred via the mediator (R, M, C), when it is transferred via the workflow engine (R, W, C), and when it is transferred via both the mediator and the workflow engine (R, M, W, C). DRC path layouts involving job and data resource nodes are not considered. Let $\mathcal{D}_P := \{(R, C), (R, M, C), (R, W, C), (R, M, W, C)\}$ be the *set of DRC path types* and let $\mathcal{D}_S = \{Pip, \neg Pip\}$ be the *set of DRC staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined DRC staging.

Definition 3.9 (DASW bulk data flow)

Similarly to DASGs, in the case of DASWs first bulk data have to be transferred from D to C, since this is the only entity that can communicate with D. Next bulk data is either transferred to the application directly (R, C, J), via the mediator (R, C, M, J), via the workflow engine (R, C, W, J), or via both the mediator and the workflow engine (R, C, M, W, J). Cases that transfer bulk data via the DRC repository (R) are not considered. Let $\mathcal{B}_P := \{(D, C, J), (D, C, M, J), (D, C, W, J), (D, C, M, W, J)\}$ be the *set of bulk data path types* and let $\mathcal{B}_S = \{Pip, \neg Pip\}$ be the *set of bulk data staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined bulk data staging.

Definition 3.10 (DASW data flow types)

Having these, let $\mathcal{DF} := \mathcal{D}_P \times \mathcal{D}_S \times \mathcal{B}_P \times \mathcal{B}_S$ be the *set of DASW data flow types.*

3.2.4 DASW resource

Definition 3.11 (DASW resource layout)

Since definition 3.1 identified the same core DASW node types (M, R, E), these can have exactly the same resource types as in the case of DASGs described in definition 2.34. The three external DASW node types (W, J, D) are hosted on external machines. Based on these, the set of all possible DASW resource layouts is defined over node type set $\mathcal{T}$ as:

$$\begin{aligned}\mathcal{L}_R := \{\xi \in \mathcal{L}_R(\mathcal{T}) \| \xi_M \in \{DeM, ExM\} \wedge \xi_R \neq CoM \wedge \xi_C \neq StM \wedge \\ \wedge \xi_W, \xi_J, \xi_D = ExM\}\end{aligned} \tag{3.1}$$

3.2.5 DASW interface

Definition 3.12 (DASW Interfaces)

Let $\mathcal{I}_F := \{Gen, Spe\}$ be the *set of frontend interface types*, let $\mathcal{I}_B := \{CLI, API\}$ be the *set of backend interface types*, and let $\mathcal{IN} := \mathcal{I}_F \times \mathcal{I}_B$ be the *set of interface types*.

3.2.6 DASW integration

Definition 3.13 (DASW Subject of integration)

The *subject of integration* is the particular part of the workflow system that will be able to communicate with the mediator. Let $\mathcal{G}_S := \{WEd, AuT, WEn\}$ be the *set of integration subjects*. WEd represents *workflow editor level integration* that enables the workflow editor to be capable of communicating with heterogeneous data resources. AuT represents *auxiliary tool level integration* where the workflow management system is extended with an auxiliary tool that is able to access hetero-

geneous data resources. WEn represents *workflow engine level integration* meaning that the workflow engine is enhanced to be able to execute the data requests.

Definition 3.14 (Request representation)

Let $\mathcal{G}_R := \{PLR, JLR\}$ be the *set of request representation types*, where PLR indicates port level representation, meaning that the data request is represented as either an input port of a workflow job that will process the result of the request, or as an output port if the job produces data that has to be transferred to a data resource. JLR indicates job level representation, meaning that the request is represented as a workflow job that transfers it to the data resource and receives the results. See figure 3.4.

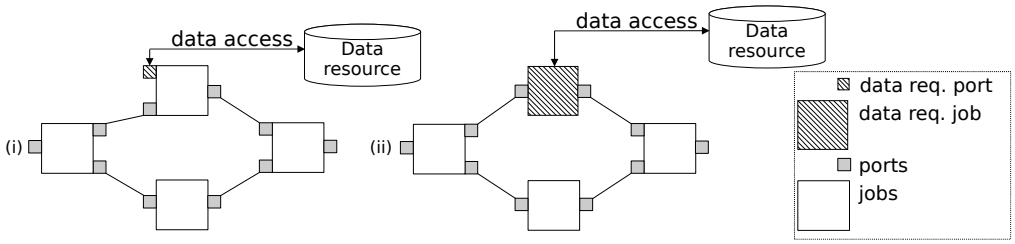

Figure 3.4: Request representation within workflows: (i) port level, (ii) job level

Definition 3.15 (Set of possible DASW integrations)

The set of possible DASW integrations is defined as $\mathcal{IG} := \mathcal{G}_S \times \mathcal{G}_R$.

3.2.7 DASW architecture and solution

Definition 3.16 (Set of DASW architectures)
The *set of DASW architectures* can be composed as:

$$\begin{aligned}\mathcal{AR} := \{((h,\delta),\xi,(q_d,s_d,c_d,q_b,s_b),(i_f,i_b),(g_s,g_r)) \in \mathcal{L}_S \times \mathcal{L}_R \times \mathcal{DF} \times \mathcal{IN} \times \mathcal{IG} \| \\ q_d \text{ and } q_b \text{ are acyclic path layouts based on instance layout } h \wedge \quad (i) \\ \wedge \forall (t,D) \in h : t = D \wedge \quad (ii) \\ \wedge \forall (t_1,t_2) \in h : \xi_{t_1} = \xi_{t_2}\}. \quad (iii)\end{aligned} \tag{3.2}$$

Note that conditions are needed for the same reasons as described in the case of DASG architectures in definition 2.40.

Definition 3.17 (DASW solution)
A DASW solution is a set of DASW architectures. With other words, it is a not empty subset of $\mathcal{AR}$.

3.3 DASW architecture analysis

3.3.1 DASW generality, extendibility, and data access

For the same reasons described in the case of DASGs in section 2.3.1, general frontend interface, command line backend interface, and centralised mediator hosted on a dedicated machine are recommended.

In terms of subject of integration, workflow editor level integration means that the scientist can gather the data before workflow execution as part of the workflow

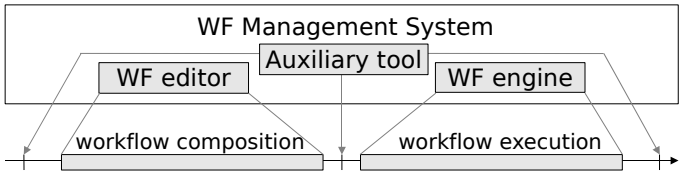

Figure 3.5: Subject of integration

design process. Data requests are executed at authoring time, the results of the request will be part of the concrete executable workflow. Since this solution provides access only before workflow execution, it supports only static data staging. Auxiliary tool level integration provides data access before and after workflow execution as an individual part of the system. (For instance, in the case of a portal-based workflow management system this tool can be a portlet.) Similarly to workflow editor level integration, this solution supports only static data staging. Data-sets are transferred between the data resource and a storage which the concrete workflow is able to access. Data is delivered from the database to the workflow by the auxiliary tool. Then, the workflow processes the data and generates an output as the result of the computation. This result-set can be transferred back to the resource by the tool. Data requests are separated from the workflow, the only connection is the shared storage that both the workflow engine and the auxiliary tool can access. Workflow engine level integration provides runtime access to the data resource. Contrary to workflow editor and auxiliary tool level integration, this solution supports semi-dynamic and dynamic data staging. Therefore, it is recommended to integrate DASW with the workflow engine of a workflow management system. Figure 3.5 illustrates, using a workflow life-cycle time-line, when the different subject of integration approaches enable data access.

In the case of both port and node level request representation, the parameters of a data request can be generated by previous nodes of the workflow at runtime. Therefore, both approaches support dynamic data staging. However, port-level

representation approach is best applied if the data request is nothing more than a simple data transfer, while, node-level representation is best applied when the data request is more complex. Since, in general data requests can contain complex queries, data transformation, and even computation, it is recommended to represent data requests at node level.

3.3.2 DASW performance

The performance analysis compares different DASW architectures focusing on overhead, latency, overhead scalability and latency scalability. It is based on DASW scenarios where $r \in \mathbb{N}^+$ different workflow jobs hosted by r different machines gather data of equal size from r decoupled data resources via a particular DASW architecture simultaneously. Each job is part of a different workflow that is executed by engines hosted on r different machines. These scenarios are represented as the elements of the set defined below.

Definition 3.18 (DASW scenarios)

$$
\begin{aligned}
\text{Let } \mathcal{AS} := \{((h,\delta),\xi,q_d,q_b,w_d,s_d,s_b,l_d,l_b,r) \in \mathcal{L}_S \times \mathcal{L}_R \times \mathcal{D}_P \times \mathcal{B}_P \times \mathbb{R}_0^+ \times (\mathbb{N}^+)^5 \| \\
\forall t \in \mathcal{T}'' : \delta_t = r \wedge \quad (i) \\
\wedge q_d \text{ and } q_b \text{ are acyclic path layouts based on } h \wedge \quad (ii) \\
\wedge \forall (t,D) \in h : t = D \wedge \quad (iii) \\
\wedge \forall (t_1,t_2) \in h : \xi_{t_1} = \xi_{t_2} \wedge \quad (iv) \\
\wedge \xi_C \neq CoM \Rightarrow w_d = 0 \wedge \quad (v) \\
\wedge \forall t \in \mathcal{T}' : r \equiv 0 \mod \delta_t \wedge \quad (vi) \\
\wedge l_d \equiv 0 \mod s_d \wedge l_b \equiv 0 \mod s_b\}. \quad (vii)
\end{aligned} \tag{3.3}
$$

be the *set of analytical DASW scenarios*. Let $a = ((h, \delta), \xi, q_d, q_b, w_d, s_d, s_b, l_d, l_b, r) \in \mathcal{AS}$ be an analytical scenario. Parameters of a and conditions of AS represent the same concepts as in the case of analytical DASG scenarios defined in definition 2.42.

Definition 3.19 (DASW scenario execution)
Since control flow is excluded from the model, similarly to the case of DASG scenario execution, the analysis is based on DRC and bulk data flow. $\forall i \in [1..r]$: let $d_i \in \mathcal{B}$ byte array represent the code of the DRC that communicates with D_i and to be transferred from R_i to C_i and $b_i \in \mathcal{B}$ byte array represent the bulk data that is to be transferred from D_i to J_i. A DASW scenario is executed in three steps:

1. DRC transfer: $\forall i \in [1..r]$: d_i is transferred from R_i to C_i via path $\psi_i(q_d)$ simultaneously.
2. DRC queuing: all DRCs are waiting w_d amount of time to be scheduled for execution,
3. Bulk data transfer: $\forall i \in [1..r]$: the execution of the ith DRC starts and b_i is transferred from D_i to J_i via path $\psi_i(q_b)$ simultaneously.

DRC flow can be represented as a simultaneous transfer (see definition 2.48), since the conditions of definition 3.18 ensure that $((h, \delta), q_d, s_d, l_d, r) \in \mathcal{D}_{st}$. Similarly, bulk data flow also can be represented as: $((h, \delta), q_b, s_b, l_b, r) \in \mathcal{D}_{st}$. Having these, the performance functions of a DASW scenario can be defined as follows.

Definition 3.20 (Performance function of DASW DRC transfer)
Let $a := ((h, \delta), \xi, q_d, q_b, w_d, s_d, s_b, l_d, l_b, r) \in \mathcal{AS}$, and $\Gamma_d : \mathcal{AS} \to \mathbb{R}_0^+$ be a function for determining execution time of DRC transfer as:

$$\Gamma_d(a) := \tau_q((h, \delta), q_d, s_d, l_d, r) \tag{3.4}$$

Definition 3.21 (Performance functions of DASW bulk data transfer)
Let $a := ((h, \delta), \xi, q_d, q_b, w_d, s_d, s_b, l_d, l_b, r) \in \mathcal{AS}$, and $\Gamma_b, \Delta_b, \Theta_b : \mathcal{AS} \to \mathbb{R}_0^+$ be functions for determining respectively bulk data transfer time, overhead, and latency as:

$$\Gamma_b(a) := \tau_q((h, \delta), q_b, s_b, l_b, r), \tag{3.5}$$

$$\Delta_b(a) := \Gamma_b(a) - k_b s_b K, \text{ and} \tag{3.6}$$

$$\Theta_b(a) := \epsilon_q((h, \delta), q_b, s_b, l_b, r). \tag{3.7}$$

Note that definition 2.49 is applied to determine $\Gamma_b(a)$ and $\Theta_b(a)$. $\forall i \in [1..r]$: transferring b_i directly between D_i and J_i takes $\tau_e(\lambda(b_i), (D_i, J_i)) = k_b s_b K$ time. Overhead on bulk data transfer of a particular scenario is considered as this time subtracted from bulk data transfer time.

Definition 3.22 (Overall DASW performance functions)
Let $\Gamma, \Delta, \Theta : \mathcal{AS} \to \mathbb{R}_0^+$ be functions for determining respectively overall execution time, overhead, and latency of a scenario, where:

$$\Gamma(a) := w_d + \Gamma_d(a) + \Gamma_b(a), \tag{3.8}$$

$$\Delta(a) := w_d + \Gamma_d(a) + \Delta_b(a), \text{ and} \tag{3.9}$$

$$\Theta(a) := w_d + \Gamma_d(a) + \Theta_b(a). \tag{3.10}$$

Because DRC transfer always has to be performed before bulk data transfer, DRC transfer time is always added to the latency and overhead of a scenario.

Definition 3.23 (Scalability of DASW data transfer)
Performance functions of any DASW scenario are characterised based on growth rates in function of l_b and r. This is represented by the Bachmann–Landau (Big O) notation and can have the same values as in the case of DASGs described in definition 2.53.

3.3.3 DASW bulk data flow

Case	Bulk data path	Instance layout	Restrictions	Group
BC1 *	(D, C, J)	$(C, J) \in h$	$\delta_C = r$	BG1
BC2	(D, C, M, J)	$(C, M), (M, J) \in h$	$\delta_M, \delta_C = r$	
BC3	(D, C, W, J)	$(C, W), (W, J) \in h$	$\delta_C = r$	
BC4	(D, C, M, W, J)	$(C, M), (M, W), (W, J) \in h$	$\delta_M, \delta_C = r$	
BC5 *	(D, C, J)	$(C, J) \notin h$		BG2
BC6	(D, C, M, J)	$(C, M) \notin h \wedge (M, J) \in h$	$\delta_M = r$	
BC7	(D, C, M, J)	$(C, M) \in h \wedge (M, J) \notin h$	$\delta_M = \delta_C$	
BC8	(D, C, W, J)	$(C, W) \notin h \wedge (J, W) \in h$		
BC9	(D, C, W, J)	$(C, W) \in h \wedge (W, J) \notin h$	$\delta_C = r$	
BC10	(D, C, M, W, J)	$(C, M), (M, W) \in h \wedge (W, J) \notin h$	$\delta_M = \delta_C$	
BC11	(D, C, M, W, J)	$(C, M) \notin h \wedge (M, W), (W, J) \in h$	$\delta_M = r$	
BC12	(D, C, M, W, J)	$(M, W) \notin h \wedge (C, M), (W, J) \in h$	$\delta_M, \delta_C = r$	
BC13 *	(D, C, M, J)	$(C, M), (M, J) \notin h$		BG3
BC14	(D, C, M, W, J)	$(C, M), (M, W) \notin h \wedge (W, J) \in h$		
BC15	(D, C, M, W, J)	$(C, M), (W, J) \notin h \wedge (M, W) \in h$	$\delta_M = r$	BG3 $\wedge$ BG4
BC16 *	(D, C, W, J)	$(C, W), (W, J) \notin h$		BG4
BC17	(D, C, M, W, J)	$(C, M) \in h \wedge (M, W), (W, J) \notin h$	$\delta_M = \delta_C$	
BC18 *	(D, C, M, W, J)	$(C, M), (M, W), (W, J) \notin h$		BG5

Table 3.2: DASW of bulk data flow cases

There are 4 bulk data path layouts: (D, C, J), (D, C, M, J), (D, C, W, J), and (D, C, M, W, J). Based on these 36 different bulk data flow cases can be identified (see definition 2.54). However, since DASW instance layouts having (D, C) coupled are not considered, $\pi(D, C)$ should always be false. This means that only 18 data flow cases are considered. These are listed in table 3.2, where representative cases (see definition 2.54) are marked with asterisks. Based on definition 2.15: $\forall(t_1, t_2) \in h : t_1, t_2 \in \mathcal{T} \Rightarrow \delta_{t_1} = \delta_{t_2}$. This means that instance layout implies restrictions of type layout. This is also included in the table. Based on definition 2.55, bulk data flow cases can be divided into 5 different groups, where cases of each group are equivalent, in terms that they have the same performance characteristics. These groups are also illustrated in the table 3.2.

Group	Overhead ($\Delta_b(a)$)	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
BG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG2	$\frac{r(k_b+1)s_bK}{\min\{r,\delta_C\}} - k_b s_b K$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$	$k_b = 1$ $\delta_C \geqq r$	$\nexists$	$\delta_C < r$
BG3	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rs_bK}{\min\{r,\delta_C\}} +$ $+\frac{rk_bs_bK}{\min\{r,\delta_M,\delta_C\}} - k_b s_b K$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$k_b = 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$\nexists$	$\delta_C < r \vee \delta_M < r$
BG4	$s_bK + \frac{r(k_b+1)s_bK}{\min\{r,\delta_C\}} - k_b s_b K$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$	$k_b = 1$ $\delta_C \geqq r$	$\nexists$	$\delta_C < r$
BG5	$s_bK + \frac{rs_bK}{\min\{r,\delta_C\}} + \frac{rs_bK}{\min\{r,\delta_M\}} +$ $+\frac{rk_bs_bK}{\min\{r,\delta_C,\delta_M\}} - k_b s_b K$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$k_b = 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$\nexists$	$\delta_C < r \vee \delta_M < r$

Table 3.3: Overhead and scalability of DASW bulk data staging

Group	Latency ($\Theta_b(a)$)	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
BG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG2	$\frac{rs_bK}{\min\{r,\delta_C\}}$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$	$k_b = 1$ $\delta_C \geqq r$	$k_b > 1$ $\delta_C < r$	$k_b = 1$ $\delta_C < r$
BG3	$\frac{rs_bK}{\min\{r,\delta_M\}} +$ $+\frac{rs_bK}{\min\{r,\delta_C\}}$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$k_b = 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$k_b > 1$ $\delta_C < r \vee \delta_M < r$	$k_b = 1$ $\delta_C < r \vee \delta_M < r$
BG4	$s_bK + \frac{rs_bK}{\min\{r,\delta_C\}}$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$	$k_b = 1$ $\delta_C \geqq r$	$k_b > 1$ $\delta_C < r$	$k_b = 1$ $\delta_C < r$
BG5	$\frac{rs_bK}{\min\{r,\delta_C\}} +$ $+\frac{rs_bK}{\min\{r,\delta_C,\delta_M\}} +$ $+\frac{rs_bK}{\min\{r,\delta_M\}}$	$\nexists$	$k_b > 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$k_b = 1$ $\delta_C \geqq r$ $\delta_M \geqq r$	$k_b > 1$ $\delta_C < r \vee \delta_M < r$	$k_b = 1$ $\delta_C < r \vee \delta_M < r$

Table 3.4: Latency and scalability of DASW bulk data staging

Transfer time, overhead, and latency values can be found in table 3.5, 3.3, and 3.4 respectively, where the formulas are based on the definition of $\Gamma_b(a)$, $\Delta_b(a)$, and $\Theta_b(a)$. Based on $\Delta_b(a)$ and $\Theta_b(a)$, the architectural conditions which determine scalability in terms of overhead and latency are identified for each group, these can be found in table 3.3, and 3.4. In particular, cases of group BG1 always provide 0 overhead and latency on bulk data staging, while cases of group BG2, BG3, BG4, and BG5 never. In the cases of the latter four groups the same rules apply as in the case of DASG bulk data transfer groups BG2 and BG3 described in section 2.3.3.

Group	Transfer time ($\Gamma_b(a)$)
BG1	$k_b s_b K$
BG2	$\frac{r(k_b+1)s_b K}{\min\{r,\delta_C\}}$
BG3	$\frac{rs_b K}{\min\{r,\delta_M\}} + \frac{rs_b K}{\min\{r,\delta_C\}} + \frac{rk_b s_b K}{\min\{r,\delta_M,\delta_C\}}$
BG4	$s_b K + \frac{r(k_b+1)s_b K}{\min\{r,\delta_C\}}$
BG5	$s_b K + \frac{rs_b K}{\min\{r,\delta_C\}} + \frac{rs_b K}{\min\{r,\delta_M\}} + \frac{rk_b s_b K}{\min\{r,\delta_C,\delta_M\}}$

Table 3.5: Time of DASW bulk data transfer

3.3.4 DASW DRC flow

Case	DRC path	Instance layout	Restrictions	Group
DC1 *	(R,C)	$(R,C) \in h$	$\delta_R = \delta_C$	DG1
DC2	(R,M,C)	$(R,M),(M,C) \in h$	$\delta_M = \delta_R = \delta_C$	
DC3	(R,W,C)	$(R,W),(W,C) \in h$	$\delta_R, \delta_C = r$	
DC4	(R,M,W,C)	$(R,M),(M,W),(W,C) \in h$	$\delta_M, \delta_R, \delta_C = r$	
DC5 *	(R,C)	$(R,C) \notin h$		DG2
DC6	(R,M,C)	$(R,M) \in h \wedge (M,C) \notin h$	$\delta_M = \delta_R$	
DC7	(R,M,C)	$(R,M) \notin h \wedge (M,C) \in h$	$\delta_M = \delta_C$	
DC8	(R,W,C)	$(R,W) \in h \wedge (W,C) \notin h$	$\delta_M = r$	
DC9	(R,W,C)	$(R,W) \notin h \wedge (W,C) \in h$	$\delta_C = r$	
DC10	(R,M,W,C)	$(R,M),(M,W) \in h \wedge$ $\wedge (W,C) \notin h$	$\delta_M, \delta_R = r$	
DC11	(R,M,W,C)	$(W,C),(R,M) \in h \wedge$ $\wedge (M,W) \notin h$	$\delta_C = r$ $\delta_M = \delta_R$	
DC12	(R,M,W,C)	$(M,W),(W,C) \in h \wedge$ $\wedge (R,M) \notin h$	$\delta_M, \delta_C = r$	
DC13 *	(R,M,C)	$(R,M),(M,C) \notin h$		DG3
DC14	(R,M,W,C)	$(W,C) \in h \wedge$ $\wedge (R,M),(M,W) \notin h$	$\delta_C = r$	
DC15	(R,M,W,C)	$(M,W) \in h \wedge$ $\wedge (R,M),(W,C) \notin h$	$\delta_M = r$	DG3 $\wedge$ DG4
DC16 *	(R,W,C)	$(R,W),(W,C) \notin h$		DG4
DC17	(R,M,W,C)	$(R,M) \in h \wedge$ $\wedge (M,W),(W,C) \notin h$	$\delta_R = \delta_M$	
DC18	(R,M,W,C)	$(R,M),(M,W),(W,C) \notin h$		DG5

Table 3.6: DASW DRC data flow cases

Performance characteristics of DRC transfer are determined by DRC path, DRC

staging, instance and type layout. DRC flow analysis is similar to bulk data flow analysis in several aspects. Based on the four DRC path layouts ((R, C), (R, M, C), (R, W, C), (R, M, W, C)) 18 different cases can be identified. These are listed in table 3.6 along with the restrictions on type layout implied by definition 2.15. Transfer time values are determined based on the definition of $\Gamma_d(a)$ and included in table 3.7. Scalability is only analysed in function of r, since l_b does not affect DRC flow.

Group	Transfer time $\Gamma_d(a)$	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
DG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
DG2	$\frac{rk_d s_d K}{\min\{r,\delta_C,\delta_R\}}$	$\nexists$	$\delta_R \geqq r$ $\delta_C \geqq r$	$\nexists$	$\delta_R < r \vee \delta_C < r$	$\nexists$
DG3	$\frac{rs_d K}{\min\{r,\delta_R,\delta_M\}} +$ $+\frac{rs_d K}{\min\{r,\delta_M,\delta_C\}} +$ $+\frac{r(k_d-1)s_d K}{\min\{r,\delta_M,\delta_R,\delta_C\}}$	$\nexists$	$\delta_M \geqq r$ $\delta_C \geqq r$ $\delta_R \geqq r$	$\nexists$	$\delta_M < r \vee \delta_R < r \vee \delta_C < r$	$\nexists$
DG4	$\frac{rs_d K}{\min\{r,\delta_R\}} +$ $+\frac{rs_d K}{\min\{r,\delta_C\}} +$ $+\frac{r(k_d-1)s_d K}{\min\{r,\delta_R,\delta_C\}}$	$\nexists$	$\delta_C \geqq r$ $\delta_R \geqq r$	$\nexists$	$\delta_R < r \vee \delta_C < r$	$\nexists$
DG5	$\frac{rs_d K}{\min\{r,\delta_R,\delta_M\}} +$ $+\frac{rs_d K}{\min\{r,\delta_M\}} +$ $+\frac{rs_d K}{\min\{r,\delta_C\}} +$ $+\frac{r(k_d-1)s_d K}{\min\{r,\delta_M,\delta_R,\delta_C\}}$	$\nexists$	$\delta_M \geqq r$ $\delta_C \geqq r$ $\delta_R \geqq r$	$\nexists$	$\delta_M < r \vee \delta_R < r \vee \delta_C < r$	$\nexists$

Table 3.7: Transfer time and scalability of DASW DRC transfer

Cases of group DG1 require, that all nodes through which DRCs are transferred are hosted by the same machine, that implies that DRC transfer time is 0 and scalability is $O(0)$. In cases of group DG2, DG3, DG4, and DG5 the same rules apply as in the case of DASG DRC transfer groups DG2 and DG3 described in section 2.3.4. Transfer time and scalability values are listed in table 3.7.

3.3.5 Recommended DASW structure layout, data flow, and resource layout

In any of the 18 bulk data transfer cases, it is possible to realise scalable bulk data transfer where overhead and latency on bulk data transfer are independent of r and l_b. In terms of type layout, while cases BC1, BC3, BC5, BC8, BC9, and BC16 require that $\delta_C \geqq r$, cases BC4, BC2, BC6, BC7, BC10, BC11, BC12, BC13, BC14, BC15, BC17, and BC18 require that both $\delta_M, \delta_C \geqq r$. In terms of bulk data staging BC5 - BC18 require pipelined staging ($k_b > 1$) as well in order to provide scalable bulk data transfer.

In all of the 18 DRC transfer cases DRC transfer time is independent of l_b and in any of the 18 cases it is possible to realise scalable DCR transfer where DRC transfer time is independent of r. In terms of type layout, DC1, DC3, DC5, DC9, DC14, DC16 require that $\delta_C \geqq r$ and $\delta_R \geqq r$; DC2, DC4, DC6, DC7, DC8, DC10, DC11, DC12, DC13, DC15, DC17, and DC18 require that $\delta_C, \delta_R, \delta_M \geqq r$ to provide DRC transfer independent of r. The selection of proposed structure layout and data flow combinations is based on the following recommendations:

R1 - Mediator The same reasons apply here as in the case of DASG recommendation R1 described in section 2.3.5. Therefore, it is aimed to minimise the number of utilised dedicated machines (let $\delta_M = 1$), provide that $\delta_C \geqq r$, and exclude bulk data flow cases that transfer data via the mediator. This means that bulk data flow cases based on path type (D, C, M, J) or (D, C, M, W, J) and instance layouts having (M, C) coupled are not recommended. Furthermore, in each scenario $\delta_W, \delta_J = r$, which implies that instance layouts having (J, M) or (W, M) coupled are not recommended either.

R2 - DRC repository It cannot be guaranteed in general that DRC repositories can be hosted by the same machines that host the workflow engine or the workflow job. Therefore, instance layouts requiring to have (W, R) or (J, R) coupled are not recommended. This also means that DRC repository nodes should not be hosted on external machines ($\xi_R \neq ExM$). On the one hand, the fact that having (W, R) or (J, R) coupled are not recommended, implies that if $(W, C) \in h$ or $(J, C) \in h$, then $(R, C) \in h$ is not recommended either, because of the transitivity the of h. On the other hand, if $(W, C), (J, C) \notin h$, then (R, C) can be realised only by utilising dedicated machines for running nodes of R and C for the same reasons described in DASG recommendation R2 in section 2.3.5. Therefore, $(R, C) \in h$ in general is not recommended.

Recommendation	DASW data flow case
R1	BC2, BC4, BC6, BC7, BC10-BC15, BC17, BC18, DC2, DC4, DC7, DC10, DC12, DC15
R2	DC1-DC4, DC8
R4	DC7, DC12-DC15, DC18

Table 3.8: Elimination of DASW data flow cases based on different recommendations.

R3 - DRC execution Since $(M, C) \in h$ is not recommended based on R1, the DRC is either executed on a separate computational machine (C is decoupled), runs on the same machine as the workflow job ($(J, C) \in h$), or runs on the same machine as the workflow engine ($(W, C) \in h$). In the latter two cases, $\xi_C = ExM$. In the former case, the DRC can be executed either on a computational or on a dedicated machine. However, the number of dedicated machines should be minimised and $\delta_C \geqq r$ is aimed to be provided based on R1. Therefore, $\xi_C = DeM$ is not recommended in general.

R4 - DRC transfer Based on the reasons described in the case of DASG recommendation R4 (section 2.3.5), if $(R, M) \notin h$, then only DRC path type (R, C) or $\{(R, W), (W, C)\}$ can be recommended.

Based on these recommendations, data flow cases listed in table 3.8 are excluded. This means, that only bulk data flow case BC1, BC3, BC5, BC8, BC9, BC16 and DRC flow case DC5, DC6, DC9, DC11, DC16, DC17 can be recommended. Table 3.9 defines all possible structure layout, data flow and resource layout combinations. These can be recommended in different cases.

Aspects			Proposed													
			Proposed 1 (PC1)	Proposed 2 (PC2)	Proposed 3 (PC3)	Proposed 4 (PC4)	Proposed 5 (PC5)	Proposed 6 (PC6)	Proposed 7 (PC7)	Proposed 8 (PC8)	Proposed 9 (PC9)	Proposed 10 (PC10)	Proposed 11 (PC11)	Proposed 12 (PC12)	Proposed 13 (PC13)	Proposed 14 (PC14)
Structure	Instance layout	(W,M)														
		(W,R)														
		(W,C)	O*			X				O*			X			
		(J,C)	X	X						X	X					
		(M,R)								X	X	X	X	X	X	X
		(M,C)														
		(R,C)														
	Type layout	δ_M	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		δ_R	≥1	≥1	≥1	≥1	≥1	≥1	≥1	1	1	1	1	1	1	1
		δ_C	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r
Data flow	Bulk data path	(D,C,J)	X	X	X	O	X			X	X	X	O	X		
		(D,C,M,J)														
		(D,C,W,J)	O*			O		X	X	O*			O		X	X
		(D,C,M,W,J)														
	Bulk data staging	Pipelined (k_b=1)	O	O	X	X	X	X	X	O	O	X	X	X	X	X
		Not pipelined (k_b>1)	O	O						O	O					
	DRC path	(R,C)	X		X	O		X		O		O	O		O	
		(R,M,C)								O		O	O		O	
		(R,W,C)	O*	X		O	X		X	O*	O		O	O		O
		(R,M,W,C)								O*	O		O	O		O
	DRC staging	Pipelined (k_d>1)	O	X	O	O	X	O	X	O	X	O	O	X	O	X
		Not pipelined (k_d=1)	O		O	O		O		O		O	O		O	
Resource	Resource layout	Mediator	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM
		DRC Repository	StM	StM	StM	StM	StM	StM	StM	DeM	DeM	DeM	DeM	DeM	DeM	DeM
		DRC execution	ExM	ExM	CoM	ExM	CoM	CoM	CoM	ExM	ExM	CoM	ExM	CoM	CoM	CoM

Table 3.9: Proposed DASW structures, data flows, and resource layouts, where sign X within instance layout show which nodes are coupled in a particular case. Sign X within data flow means that only the correlated architectural approach can be implemented. Sign O indicates that it is irrelevant which approach is chosen and sign O* means that if $(W, J) \in h$ is provided, then it is irrelevant which approach is chosen.

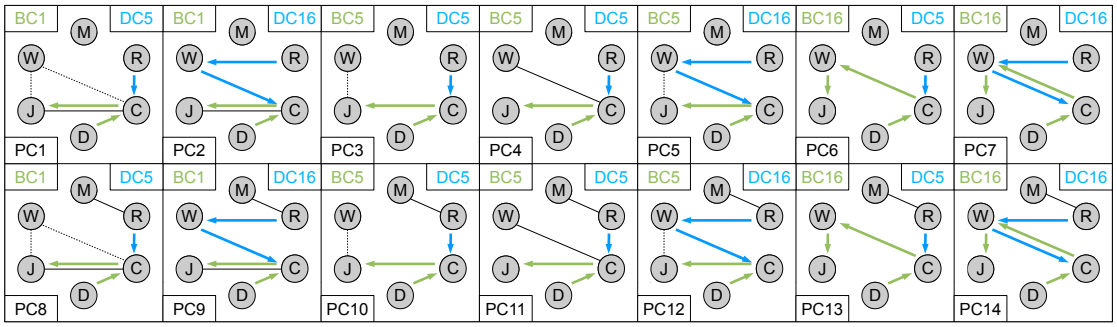

Figure 3.6: Combinations of recommended DASW data flow cases, where black lines represent which node types are coupled, dashed lines represent optional couplings, green arrows represent bulk data path layouts, and blue arrows represent DRC path layouts.

1. DRC repositories

- **(a)** If it is not possible to utilise Grid storage resources for DRC repositories then it is recommended to host the DRCs at the dedicated machine that hosts the mediator ($(M, R) \in h, \delta_R = 1, \xi_R = DeM$), since the mediator will not bottleneck DRC transfer. In this case DC5, DC6, DC9, DC11, DC16, and DC17 all can be recommended.

- **(b)** If it is possible to utilise multiple storage machines for DRC repositories then it is recommended to have a decoupled mediator ($(M, R) \notin h$ and $\delta_R \geqq 1, \xi_R = StM$). In this case only DC5, DC9, and DC16 can be recommended.

2. Workflow engines, jobs, and DRCs

- **(a)** If workflow jobs are executed locally to the workflow engines, then $(W, J) \in h$. In this case BC9 and BC16 cannot be realised, since these data flow cases require $(W, J) \notin h$. BC1, BC3, BC5, and BC8 all can be recommended.

(a-i) If the workflow engine can execute the desired DRC locally ($(W, C) \in h$), then bulk data flow case BC1 and BC3 can be recommended.

(a-ii) If $(W, C) \in h$ cannot be realised, then bulk data flow case BC5 and BC8 can be recommended.

- **(b)** If workflow jobs are executed remotely to the workflow engines ($(W, J) \notin h$), but it is possible to run the DRC on the same machine where the job runs ($(J, C) \in h$), then BC1 is recommended, since this bulk data transfer case provides 0 overhead and latency. In this case $\xi_C = ExM$.

- **(c)** If workflow jobs are executed remotely to the workflow engines ($(W, J) \notin h$) and it is not possible to run the DRC on the same machine where the job runs ($(J, C) \notin h$), then there are three options.

  (c-i) If the workflow engine can execute the desired DRC locally ($(W, C) \in h$), then data flow case BC5 or BC9 can be recommended. In this case bulk data is first transferred to the engine machine (where the DRC runs) and then to the job machine.

  (c-ii) If $(W, C) \in h$ cannot be realised and data can be transferred directly to the machine that runs the job from the machine that runs the DRC, then bulk data flow case BC5 can be recommended. In this case bulk data is first transferred to the separate machine that runs the DRC and then to the job machine.

  (c-iii) If $(W, C) \in h$ cannot be realised and data cannot be transferred directly to the machine that runs the job from the machine that runs the DRC, then bulk data flow case BC16 is recommended. In this case bulk data is first transferred to the separate machine that runs the DRC, next to the engine machine and finally to the job machine.

Based on these table 3.10 summarises which data flow and structure layout com-

Case	Proposed combinations
1/a, 2/a-i	PC8
1/a, 2/a-ii	PC10, PC12
1/a, 2/b	PC8, PC9
1/a, 2/c-i	PC11
1/a, 2/c-ii	PC10, PC12
1/a, 2/c-iii	PC13, PC14

Case	Proposed combinations
1/b 2/a-i	PC1,
1/b 2/a-ii	PC3, PC5
1/b, 2/b	PC1, PC2
1/b, 2/c-i	PC4
1/b, 2/c-ii	PC3, PC5
1/b, 2/c-iii	PC6, PC7

Table 3.10: Proposed DASW structure and data flow combinations in different cases

binations are recommended in the different cases. Table 3.11 illustrates performance characteristics of each proposed combination.

Based on definition 2.54, BC1 is the representative of the group that also includes BC3. Note that, since BC1 and BC3 are equivalent (see definition 2.55), their performance characteristics are the same. This also applies for BC5 and BC8, BC9; DC5 and DC6, DC9, DC11; and DC16 and DC17. Rather than illustrating all combinations of each case, here only the representative cases BC1, BC5, BC16, DC5, and DC16 are illustrated in figure 3.6 with the possible structure layouts that can implement them.

3.4 Existing and proposed DASW solutions

3.4.1 Existing DASW solutions

There are several solutions that can be utilised by workflows for accessing distributed data. SAGA, SRB, LFC, Hadoop MapReduce can be used for this purpose. However, as it is explained in section 2.4.1, these solutions cannot provide access to a wide range of heterogeneous data resources of different types and vendors. For this

Proposed	Overhead / Latency	Overhead / Latency scalability
PC1	$w_d + \frac{rl_dK}{\min\{r,\delta_R\}}$	$O(r+1)$
PC2	$w_d + \frac{rl_dK}{\min\{r,\delta_R\}} + s_dK$	$O(r+1)$
PC3	$w_d + \frac{rl_dK}{\min\{r,\delta_R\}} + s_bK$	$O(r+1)$
PC4	$\frac{rl_dK}{\min\{r,\delta_R\}} + s_bK$	$O(r+1)$
PC5	$w_d + \frac{rl_dK}{\min\{r,\delta_R\}} + s_dK + s_bK$	$O(r+1)$
PC6	$w_d + \frac{rl_dK}{\min\{r,\delta_R\}} + 2s_bK$	$O(r+1)$
PC7	$w_d + \frac{rl_dK}{\min\{r,\delta_R\}} + s_dK + 2s_bK$	$O(r+1)$
PC8	$w_d + rl_dK$	$O(r+1)$
PC9	$w_d + rl_dK + s_dK$	$O(r+1)$
PC10	$w_d + rl_dK + s_bK$	$O(r+1)$
PC11	$rl_dK + s_b$	$O(r+1)$
PC12	$w_d + rl_dK + s_dK + s_bK$	$O(r+1)$
PC13	$w_d + rl_dK + 2s_bK$	$O(r+1)$
PC14	$w_d + rl_dK + s_dK + 2s_bK$	$O(r+1)$

Table 3.11: Performance characteristics of proposed DASW structures and data flows

reason they cannot be considered as DASWs.

As a matter of fact, there are no general solutions designed for workflow systems to provide access to a wide range of heterogeneous data resources. However, some workflow systems support different kinds of heterogeneous data access. These solutions are analysed in the followings.

JDBC is integrated to both Taverna and Kepler workflow systems. Data resource clients can be connected to JDBC using a backend API, while at the frontend side it provides API that is specific to SQL. Workflow engine (W), DRC repository (R), and DRC execution (C) are coupled with the mediator (M) node. Hence, these solutions provide instance layout where $h = \overline{\{(W,M),(W,R),(W,C),(M,R),(M,C),(R,C)\}}$ if jobs (J) are executed remotely to the workflow engine and $h = \overline{\{(W,M),(W,R),(W,C),(J,C),(M,R),(M,C),(R,C)\}}$ if jobs are executed locally to the workflow engine. In terms of type layout $\delta_M = \delta_C = \delta_R$, which all equal to δ_W since all

these nodes are coupled. This also means that $\xi_M = \xi_R = \xi_C = ExM$. There is no physical DRC transfer, the DRC repository is basically a set of JDBC drivers (usually represented as JAR files) that encapsulate the DRCs. Typically type 4 drivers (see comparison of different JDBC driver types in [102]) are used which are loaded and executed by the Java virtual machine and communicate directly with a particular data resource. Hence, DRC path is (R, C) and since these are coupled, DRC staging is irrelevant (see lemma A.3). Bulk data is first passed from the data resource to the the DRC (JDBC driver), this passes it further to the mediator (JDBC driver manager). Next, data is transferred to the workflow engine, that transfers it further to the job. For this reason, bulk data path is (D, C, M, W, J) and staging is not pipelined, since streaming is not supported by these solutions. JDBC is integrated with the workflow engine in the case of both Taverna and Kepler engines and data requests are represented as jobs. Requests can be generated by previous jobs, hence, data access is dynamic.

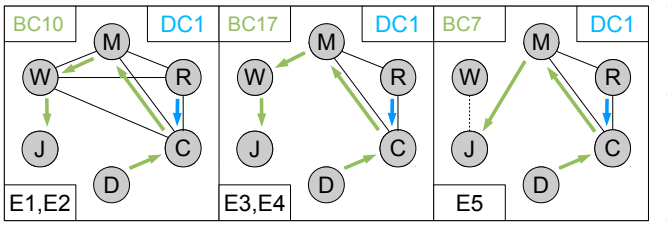

Figure 3.7: Structure and data flow cases of the existing architectures, where black lines represent which node types are coupled, green arrows represent bulk data path layouts and blue arrows represent DRC path layouts.

A proof-of-concept that integrates OGSA-DAI WSI 2.2 and Taverna 1.4 was developed by the Taverna team [103]. A new job type was introduced into Taverna called OGSA-DAI processor, that is able to execute an SQL query via a given OGSA-DAI service. Since the solution is based on OGSA-DAI, backend interface is an API and the frontend interface is general, any type of data resource can be connected

Aspects			Existing					Proposed													
Solutions			JDBC with Taverna (E1)	JDBC with Kepler (E2)	OGSA-DAI with Taverna POC (E3)	OGSA-DAI portlets with WF portals T1 (E4)	OGSA-DAI portlets with WF portals T2 (E5)	Proposed 1 (PC1)	Proposed 2 (PC2)	Proposed 3 (PC3)	Proposed 4 (PC4)	Proposed 5 (PC5)	Proposed 6 (PC6)	Proposed 7 (PC7)	Proposed 8 (PC8)	Proposed 9 (PC9)	Proposed 10 (PC10)	Proposed 11 (PC11)	Proposed 12 (PC12)	Proposed 13 (PC13)	Proposed 14 (PC14)
Structure	Instance layout	(W,M)	X	X																	
		(W,R)	X	X																	
		(W,C)	X	X				O*			X				O*			X			
		(J,C)	O*	O*				X	X						X	X					
		(M,R)	X	X	X	X	X								X	X	X	X	X	X	X
		(M,C)	X	X	X	X	X														
		(R,C)	X	X	X	X	X														
	Type layout	δ_M	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		δ_R	1	1	1	1	1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	1	1	1	1	1	1	1
		δ_C	1	1	1	1	1	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r
Data flow	Bulk data path	(D,C,J)						X	X	X	O	X			X	X	X	O	X		
		(D,C,M,J)					X														
		(D,C,W,J)						O*			O		X	X	O*			O		X	X
		(D,C,M,W,J)	X	X	X	X	X														
	Bulk data staging	Pipelined (k_s=1)						O	O	X	X	X	X	X	O	O	X	X	X	X	X
		Not pipelined (k_s>1)	X	X	X	X	X	O	O						O	O					
	DRC path	(R,C)	X	X	X	X	X	X		X	O		X		O		O	O		O	
		(R,M,C)													O		O	O		O	
		(R,W,C)						O*	X		O	X		X	O*	O		O	O		O
		(R,M,W,C)													O*	O		O	O		O
	DRC staging	Pipelined (k_s>1)	O	O	O	O	O	O	X	O	O	X	O	X	O	X	O	O	X	O	X
		Not pipelined (k_s=1)	O	O	O	O	O	O		O	O		O		O		O	O		O	
Resource	Resource layout	Mediator	ExM	ExM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM
		DRC Repository	ExM	ExM	DeM	DeM	DeM	StM	StM	StM	StM	StM	StM	StM	DeM	DeM	DeM	DeM	DeM	DeM	DeM
		DRC execution	ExM	ExM	DeM	DeM	DeM	ExM	ExM	CoM	ExM	CoM	CoM	CoM	ExM	ExM	CoM	ExM	CoM	CoM	CoM
Integration	Subject of integration	Workflow editor level																			
		Auxiliary tool level				X	X														
		Workflow engine level	X	X	X			X	X	X	X	X	X	X	X	X	X	X	X	X	X
	Request representation	Port level representation				N/A	N/A														
		Job level integration	X	X	X	N/A	N/A	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Interface	Backend	CLI						X	X	X	X	X	X	X	X	X	X	X	X	X	X
		API	X	X	X	X	X														
	Frontend	General			X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Specific	X(SQL)	X(SQL)																	

Table 3.12: Existing and proposed DASW architectures, where signs have the same purpose as in the case of table 3.9.

to it. The newly introduced OGSA-DAI processor is a local processor, that, after installation, becomes a part of the workflow system. This processor represents the OGSA-DAI client that connects to a remote OGSA-DAI server. This server encapsulates the mediator (M), the DRC repository (R), and the DRC execution (C). Hence the solution implements an instance layout where $h = \overline{\{((M,R),(M,C),(R,C)\}}$. By default the OGSA-DAI server is hosted by a dedicated single machine, hence $\delta_M = \delta_C = \delta_R = 1$ and $\xi_M = \xi_R = \xi_C = DeM$. Although OGSA-DAI supports third party delivery, this functionality is not utilised by the solution, hence bulk data always flows though the mediator and the workflow engine that is integrated with the OGSA-DAI client. Bulk data path type is (D, C, M, W, J) and data trans-

fer is not pipelined. Similarly to JDBC, DRC repository and execution are on the same machine. Hence, DRC path type is (R, C), any DRC staging is irrelevant (see lemma A.3). The subject of integration is the workflow engine, which was extended with the ability of executing the new OGSA-DAI job type. Hence, the data request is represented at job level within a workflow. Data requests can be generated and passed by previous jobs to the OGSA-DAI job. Therefore, data access is dynamic. This solution is a restricted proof-of-concept, that supports only a small subset of the functionalities provided by OGSA-DAI, it is not intended to be used in production as it is stated in the manual of the solution.

Architecture	Overhead	Overhead scalability	Latency	Latency scalability
JDBC (E1, E2)	$l_b K$	$O(l_b)$	$l_b K$	$O(l_b)$
OGSA-DAI (E3)	$2rl_b K$	$O(rl_b)$	$2rl_b K$	$O(rl_b)$
OGSA-DAI (E4)	$2rl_b K$	$O(rl_b)$	$2rl_b K$	$O(rl_b)$
OGSA-DAI (E5)	$2rl_b K - l_b K$	$O(rl_b)$	$rl_b K$	$O(rl_b)$

Table 3.13: Performance characteristics of existing DASW architectures, where jobs are executed remotely to the workflow engines ($(W, J) \notin h$)

Architecture	Overhead	Overhead scalability	Latency	Latency scalability
JDBC (E1, E2)	0	$O(0)$	0	$O(0)$
OGSA-DAI (E3)	$2rl_b K - l_b K$	$O(rl_b)$	$rl_b K$	$O(rl_b)$
OGSA-DAI (E4)	$2rl_b K - l_b K$	$O(rl_b)$	$rl_b K$	$O(rl_b)$
OGSA-DAI (E5)	$2rl_b K - l_b K$	$O(rl_b)$	$rl_b K$	$O(rl_b)$

Table 3.14: Performance characteristics of DASW existing architectures, where jobs are executed locally to the workflow engines ($(W, J) \in h$)

Several projects developed re-usable, portal-based interfaces for OGSA-DAI. For instance, the Alliance OGSA-DAI Portlet [104], the OGSA-DAI portlet developed by the Sakai VRE Demonstrator project [105, 106], or the Westminster OGSA-DAI portlet set [107, 108]. The first version of the Westminster portlet set was

developed as part of my MSc project [109]. Except for the Westminster portlet set, the mentioned portlets provide only a very limited functionality and have not reached production quality. Such a portlet can be integrated to a workflow portal and can serve as an auxiliary tool for accessing heterogeneous data. In the case of auxiliary tool integration, however, it is not possible to represent the data request within the workflow, since it is executed either before or after the workflow. Since this solution is also based on OGSA-DAI, backend interface is API and the frontend interface is general, $\xi_M = \xi_R = \xi_C = DeM$, and the structure is also the same as in the case of the OGSA-DAI - Taverna proof-of-concept solution. The only difference here is that the OGSA-DAI client is embedded to a portlet and is not integrated with the workflow engine. However, this does not imply any structural difference. For the same reason, DRC flow is also identical to the OGSA-DAI - Taverna proof-of-concept solution. However, two types of bulk data flow can be realised by this solution. Bulk data path can be (D, C, M, W, J) (T1) if the workflow engine has a data storage that can be accessed by the OGSA-DAI service (e.g. via FTP or GridFTP) or can be (D, C, M, J) (T2) if the computational resource machine that runs the job can be accessed by the OGSA-DAI. In both cases the third party delivery function of OGSA-DAI is used and data transfer is not pipelined. Although, these portlets are very useful in some cases and provide solutions for several user scenarios, they provide only static data access.

Table 3.12 describes each architecture the existing solutions can realize and table 3.13, 3.14 shows performance characteristics of each of these architectures. Figure 3.7 illustrates structure and data flow examples for each existing architecture.

3.4.2 Proposed DASW solutions

Section 3.3 identifies several proposed architectures, which also can be seen in table 3.12. Based on section 3.3.1, in the case of the proposed architectures, the recommended backend interface for accessing DRCs is CLI and frontend interface is general. The recommended subject of integration is the workflow engine and data requests should be represented at node level within the workflows.

Section 3.3.5 identifies numerous recommended structure and data flow combinations. These are defined in table 3.9. Based on the workflow repositories and the workflow jobs 8 cases were defined in this section. Table 3.10 summarises which structure and data flow combinations are recommended in the different cases. Having these, table 3.12 defines all proposed architectures along with all architectures that can be realised by the existing solutions. The recommended DASW solution should realise one or more of the proposed architectures depending on where it would be used.

3.4.3 Comparison of existing and proposed DASW solutions

The existing and proposed architectures differ in several aspects. Overall overhead and latency of the existing solutions are all linear with the number of simultaneous requests (r) and bulk data size (l_b), except for the JDBC solution. In the case of workflows of which jobs are executed locally to their workflow engines, latency and overhead of both JDBC solutions are 0. In the case of workflows of which jobs are executed remotely to their workflow engines, latency and overhead of both JDBC solutions are linear with bulk data amount and independent of the number of simultaneous requests. Although in the above case the performance of the JDBC based

solutions are significantly better then the OGSA-DAI based solutions, they are neither general (specific to SQL) nor easily extendible, since if a new data resourced is introduced, its driver (DRC) has to be added to the JDBC of each workflow engine). On the other hand, the OGSA-DAI based solutions are general and provide a vast range of functionality, but their performance is rather poor. Bulk data is always transferred via the OGSA-DAI service. This increases overhead and bottlenecks data transfer. Data transfer is not scalable, it is linear with both the number of simultaneous requests and bulk data amount.

In the case of all proposed solutions DRC execution is never coupled with the DRC repository and the mediator, it is either coupled with the job, with the the workflow engine, or decoupled. This means that the DRC has to be physically transferred from the machine that hosts the DRC repository to the DRC execution machine. This adds additional overhead as in the case of the proposed DASG solutions, but the size of the DRCs are relatively small comparing to bulk data size. Hence, all proposed architectures provide relatively low overhead and latency that is independent of bulk data size. Although it is linear with r, this does not affect bulk data, only DRC transfer.

Graphical representation of performance improvements of proposed architectures to existing architectures is provided in the case of DASGs in section 2.4.3. Since based on the above, the performance improvements of proposed DASW architectures to existing DASW architectures are similar to the DASG performance improvements, graphical representation of DASW performance improvements is not provided.

3.5 Implementation

Proposed DASW architectures are based on similar concepts to proposed DASG architectures. To show that it is possible to realise these even if data access is provided for workflows, a solution was developed for accessing heterogeneous data from P-GRADE workflows. The solution is also based on the GEMLCA (see description in section 2.5) application repository and submitter. For testing the concept of proposed DASW architectures, the same MySQL client was used as in the case of DASGs.

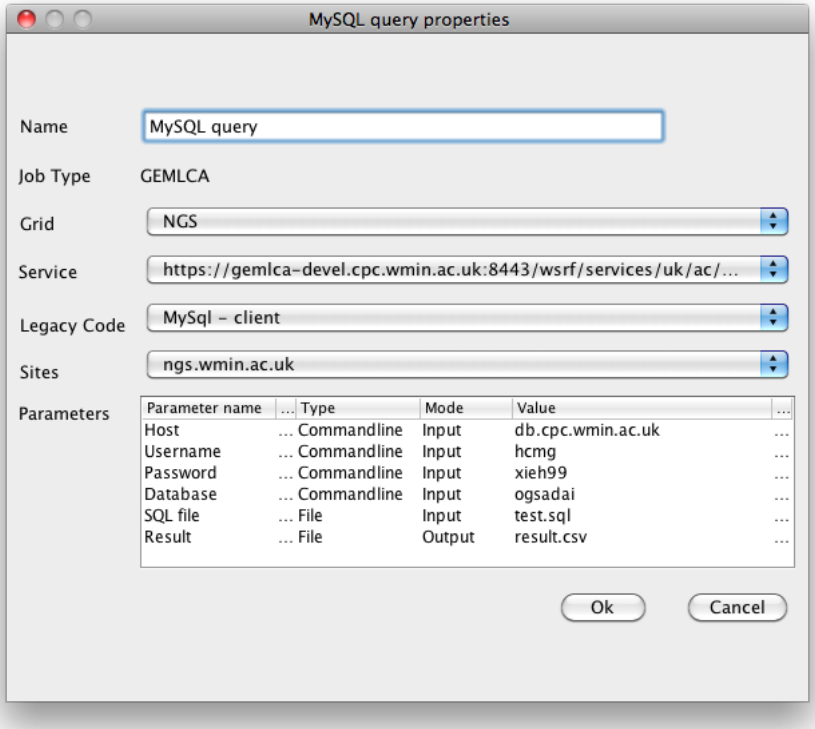

Figure 3.8: Parametrisation of a MySQL client in a P-GRADE workflow

GEMLCA is integrated [110] to the workflow engine of the P-GRADE portal in such way that a GEMLCA application is represented as a job in a P-GRADE

workflow. Figure 3.8 illustrates how this job can be parametrised in the P-GRADE workflow editor. First, the user selects the Grid and the GEMLCA service where the MySQL client is deployed. Next, selects the MySQL client from the list of available legacy applications (legacy codes) and the computational resource (site) where it will be executed. Input and output files can be generated and processed by other nodes in the P-GRADE workflow. In order to transfer bulk data directly, the MySQL client should be submitted to the same computational resource that runs the job which generates/processes it. Finally, the user parametrises the MySQL client.

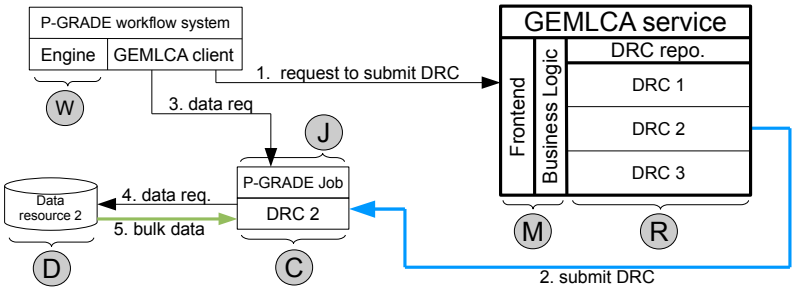

Figure 3.9: Implementation of DASW PC8 based on GEMLCA, where black arrows represent control data, blue arrow represents DRC transfer, and green arrow represents bulk data transfer.

The GEMLCA based DASW implementation is illustrated in figure 3.9. The GEMLCA application repository is used as the DRC repository, which is part of the GEMLCA service that serves as the mediator. These are hosted on the same machine, hence mediator and DRC repository are coupled. When the job which represents the data request is to be executed by the P-GRADE workflow engine, the engine passes a request to the local GEMLCA client. This request includes all information that the user specifies in the GEMLCA parameter window illustrated in figure 3.8. Next, the GEMLCA client sends a request to the selected GEMLCA service to submit the MySQL client to the desired location. When the MySQL client

is transferred, the GEMLCA client sends the data request to it. The MySQL client starts execution, sends the request to the database and retrieves the results. Bulk data flows directly between the database and the computational resource that runs the job and the MySQL client. This solution realises PC8.

3.6 Summary

Based on the mathematical model introduced in chapter 2, this chapter proposes 14 DASW architectures that provide access to heterogeneous data resources for workflows. Although many scientific experiments rely on data stored in various data resources, most workflow systems support only a small subset of data resources and many of them do not provide access to databases at all. There is a high demand for a general, easily extendible, and scalable solution that provides access to heterogeneous data resources for workflows at runtime.

Although there is no general solution designed for this purpose, some workflow systems support access to heterogeneous data resources using either JDBC or OGSA-DAI. Taverna and Kepler workflow systems support JDBC which runs on the same machine as the workflow engine. The performance of this solution is sufficient, since bulk data is not transferred via a centralised service. However, JDBC was designed for SQL based databases and cannot provide access to other types of data resources. On the other hand, if a new data resource is introduced, its driver (DRC) has to be added to the JDBC of each workflow engine.

OGSA-DAI based solutions access data using a centralised service hosted on a dedicated machine. If a new data resource is introduced, after connecting the DRC of the new data resource with OGSA-DAI, workflow systems connected with the given OGSA-DAI service can access the new data resource. The limitation of this

approach is that all data flows via the machine which hosts this centralised service. This bottlenecks data transfer and increases overhead. The result is a solution that is not suitable for large scale, data intensive workflows due to its poor performance.

The proposed architectures dynamically distribute the DRCs, which are either executed on the machine of the workflow engine, on the machine that runs the job which processes/generates bulk data, or on third party computational machines provided by the Grid, depending on the given scenario. Distribution of DRCs increases overhead, but this is minimal compared to bulk data transfer time due to the small size of the DRCs.

The reference implementation described in this chapter implements one of the proposed DASW architectures based on the P-GRADE workflow system and GEMCLA, but can be adopted by any workflow system by the integration of GEMLCA with the given system. This solution is general and easily extendible for the same reasons as the similar GEMCLA based DASG reference implementation.

Chapter 4

Heterogeneous Workflow Execution Solutions for Applications (WESA)

A WESA enables applications to execute workflows of different kinds, independently of what workflow system they were originally designed in. A WESA consists of a frontend interface, a business logic layer, a workflow repository and a backend that encapsulates multiple workflow engines. See illustration in figure 4.1. The application sends its request to the frontend. This request includes a reference to a workflow that resides in the workflow repository (or in some cases the workflow descriptor itself) and the workflow inputs and maybe some further parameters for instance to specify where the workflow should be executed. The business logic layer defines how the appropriate workflow engine is selected, parametrised and executed. The selected workflow engine executes the workflow of which output is transferred back to the application.

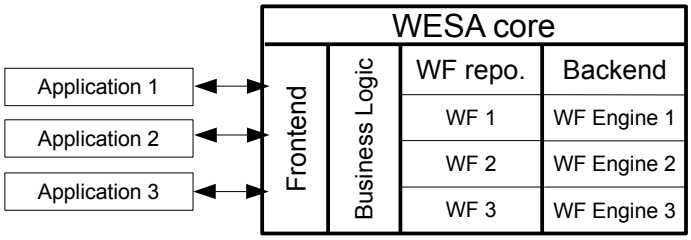

Figure 4.1: WESA concept

4.1 Key WESA properties and requirements

Although, this contribution is aiming to define an architecture for applications to access and execute heterogeneous workflows on the Grid, the same five key properties of WESAs: (i) generality, (ii) extendibility, (iii) overhead, (iv) latency, and scalability (v) were taken under consideration like in the case of DASGs described in section 2.1.

Generality Generality of a WESA architecture is defined by what kind of workflows can be executed via it. As it was discussed in section 1.3 there are numerous workflow systems, that differ in several aspects. Because of the heterogeneity of those systems, generality of such an architecture is a key property, which affects which workflow systems can be accessed via it. Although a specific metric for this property is not defined here, it is aimed to propose architectures that enable the execution of the broadest possible range of workflows.

Extendibility As well as the evolution of Grid based applications, the evolution of Grid workflow systems is dynamic. Changes in this field are so rapid it is inevitable

that systems are incompatible, and sometimes, even different versions of the same system are incompatible. Therefore, how much effort it takes to connect workflow systems to such a solution is essential. We refer to this key property as *extendibility*.

Overhead, Latency, and Scalability Furthermore, the performance related properties: *Overhead*, *latency*, and *scalability* are also important, since most scientific experiments represented and executed as Grid workflows are either or both data and computation intensive, where execution time is substantial.

4.2 WESA architecture definition

To study possible approaches and identify optimal solutions, WESAs are investigated from four aspects: *structure*, *resources*, *data flow*, and *interface*. These aspects were defined in such way that by combining them it is possible to construct several architectures. Some of them exploit more, some of them exploit less advantages provided by the underlying Grid technology. Therefore, they provide different characteristics from the key properties point of view.

Apart from the the fact that DASGs and WESAs differ in several aspects, e.g. here the mediator provides access to heterogeneous workflow engines rather than to heterogeneous data resources and the access is provided for an application in general, not for an application executed on the Grid, many of these aspects are partially identical to the aspects with the same name in section 2.2. However, as it will be described in the following description, those differences are significant enough to construct completely different architectures.

In analogy with previously defined structures, WESA architectures are also based on the general definitions defined in chapter 2.

4.2.1 WESA node

Definition 4.1 (WESA nodes and node types)

- An *application node* represents a running application that needs a workflow to be executed. In the case of WESAs this application is always executed on a single machine. Application nodes belong to type A.
- An *engine repository node* provides the executable code of a workflow engine, which is always stored locally to the engine repository node. Engine repository nodes belong to node type RE.
- A *workflow repository node* provides the workflow description of the workflow to be executed, which is always stored locally to the workflow repository node. Workflow repository nodes belong to node type RW.
- An *engine execution node* receives the executable code of a workflow engine from an engine repository node and executes it locally. After this point it represents the running workflow engine. These nodes belong to type E.
- A *job node* is task that either generates or processes data that is to be exchanged with an application node. Job nodes are initiated by workflow engine nodes and belong to node type J.
- A *mediator node* contains the WESA frontend and the business logic layer. It is contacted in order to execute a particular workflow. A mediator node performs all necessary steps in order to fulfil this request. The mediator is aware of the machines that can run the engine repository nodes, the engine execution nodes and it is also aware of the available engines. Mediator nodes belong to type M.

Note that repository nodes do not necessarily represent running services of a digital repository, they can represent any entity that is able to provide the executable

code of a workflow engine in the case of engine repositories or the workflow descriptor in the case of workflow repositories. Let $\mathcal{T}' = \{M, RE, RW, E\}$ be the set of *core WESA node types*, $\mathcal{T}'' = \{A, J\}$ be the set of *external WESA node types*, and $\mathcal{T} = \{A, J, M, RE, RW, E\}$ be the set of *all WESA node types*. See illustration on figure 4.2.

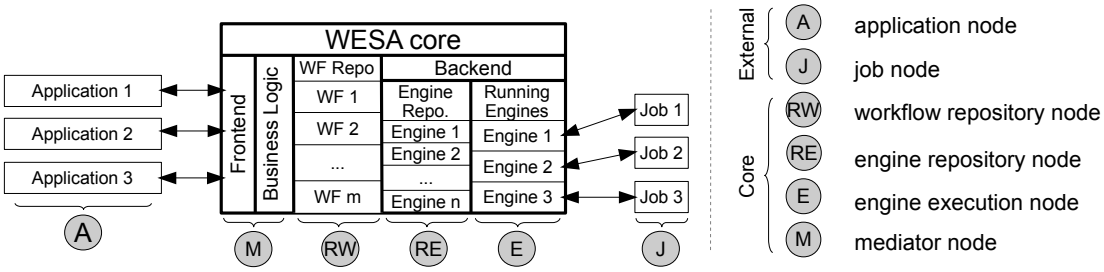

Figure 4.2: WESA node types

Definition 4.2 (WESA Instance)
Let a *WESA instance* be a set of $|\mathcal{T}| = 6$ nodes, where each node belongs to a different node type of $\{A, M, RE, RW, E, J\}$.

Definition 4.3 (Bijection between WESA node types and instances)
Let $\forall i \in [1..r]$: let $\mathcal{N}_i := \{A_i, M_i, RE_i, RW_i, E_i, J_i\}$ be the ith WESA instance, where $\varphi_i(A) = A_i$, $\varphi_i(M) = M_i$, $\varphi_i(RE) = RE_i$, $\varphi_i(RW) = RW_i$, $\varphi_i(E) = E_i$, and $\varphi_i(J) = J_i$.

In the case of WESAs there are 6 node type sets $\mathcal{N}_A$, $\mathcal{N}_M$, $\mathcal{N}_{RE}$, $\mathcal{N}_{RW}$, $\mathcal{N}_E$, $\mathcal{N}_J$ and r nodes in each type set. A WESA node matrix of instances and types can be constructed as illustrated in table 3.1. Furthermore, both $\bigcup_{i=1}^{r} \mathcal{N}_i$ and $\bigcup_{t \in \mathcal{T}} \mathcal{N}_t$ are equal to the set of all WESA nodes, $\mathcal{N}$ and $|\mathcal{N}| = 6r$.

		Node types					
		$\mathcal{N}_A$	$\mathcal{N}_M$	$\mathcal{N}_{RE}$	$\mathcal{N}_{RW}$	$\mathcal{N}_E$	$\mathcal{N}_J$
Instances	$\mathcal{N}_1$	A_1	M_1	RE_1	RW_1	E_1	J_1
	$\mathcal{N}_2$	A_2	M_2	RE_2	RW_2	E_2	J_2
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
	$\mathcal{N}_r$	A_r	M_r	RE_r	RW_r	E_r	J_r

Table 4.1: WESA node matrix

4.2.2 WESA Structure

Similarly to DASGs and DASWs, WESA structure layouts are based on two concepts: instance and type layout.

Definition 4.4 (WESA instance layout)
The set of all possible WESA instance layouts is represented by $\mathcal{L}_I$ and equals to the set of all possible type layouts on domain $\mathcal{T}$ (see definition 2.11).

Definition 4.5 (WESA type layout)
The set of all possible WESA type layouts is represented by $\mathcal{L}_T$ and equals to the set of all possible type layouts on domain $\mathcal{T}$ (see definition 2.13).

Definition 4.6 (WESA structure layout)
The set of all possible WESA structure layouts is represented by $\mathcal{L}_S$ and equals to the set of all possible structure layouts on domain $\mathcal{T}$ (see definition 2.15).

4.2.3 WESA Data flow

Definition 4.7 (WESA data types)
Data between distributed nodes of $\mathcal{N}$ can flow in various ways. To identify the different possibilities, four kinds of data are distinguished:

- *bulk data* is the data-set that needs to be transferred between (A_i) and (J_i);
- *engine data* is the workflow engine executable itself that needs to be transferred from (RE_i) to (E_i);
- *workflow data* is the workflow descriptor and all further data (fix parameters, job executables, etc) that need to be transferred from (RW_i) to (E_i); and
- *control data* is the set of information that includes all further data transferred between the nodes. The latter consists of a small number of requests which are necessary to exchange in order to enable an application to execute a workflow.

The amount of control data is typically measured in kilobytes, whilst workflow data is measured in kilo or megabytes (depending on workflow type), engine data in megabytes (see examples of the size of different engines in table 4.2) and bulk data in giga- or terabytes. The way bulk data is transferred is critical. In comparison to this, engine data flow and workflow data flow slightly, control flow barely affect the overall performance of a particular WESA architecture. Hence, only three kinds of data flow are considered here: engine data flow, workflow data flow and bulk data flow.

Definition 4.8 (WESA engine data flow)
Two engine path types are distinguished in the case of WESA, when the engine is transferred directly (RE, E) and when it is transferred via the mediator (RE, M, E). Let $\mathcal{E}_P := \{(RE, E),\ (RE, M, E)\}$ be the *set of engine path types* and let $\mathcal{E}_S =$

Workflow engine	Version	Minimal installation size (MB)	Additional modules	Overall size of additional modules (MB)
ASKALON	EE2	76.8	-	-
GWES	2.1	212.7	Linuxtoolbox, GraphViz	11.8
MOTEUR	0.9.9	21.8	GraphViz, gLiteUI, DIET API, Antlworks	14.8
Triana	3.2.3	100.7	WS Module, PEGASUS Module	29
WS-PGRADE	3.2	326	gLiteUI	10

Table 4.2: Uncompressed size of example workflow engines and optional modules

$\{Pip, \neg Pip\}$ be the *set of engine staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined engine staging.

Definition 4.9 (WESA workflow data flow)

Two engine path types are distinguished in the case of WESAs, when the engine is transferred directly (RW, E) and when it is transferred via the mediator (RW, M, E). Let $\mathcal{W}_P := \{(RW, E), (RW, M, E)\}$ be the *set of workflow path types* and let $\mathcal{W}_S = \{Pip, \neg Pip\}$ be the *set of engine staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined workflow staging.

Definition 4.10 (WESA bulk data flow)

Four bulk data path types are distinguished in the case of WESAs, when bulk data is transferred via directly (A, J), via the mediator (A, M, J), via the engine (A, E, J), and via both the mediator and the engine (A, M, E, J). Let $\mathcal{B}_P := \{(A, J), (A, M, J), (A, E, J), (A, M, E, J)\}$ be the *set of bulk data path types* and let $\mathcal{B}_S = \{Pip, \neg Pip\}$ be the *set of bulk data staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined bulk data staging.

Definition 4.11 (WESA data flow types)

Having these, let $\mathcal{DF} := \mathcal{E}_P \times \mathcal{E}_S \times \mathcal{W}_P \times \mathcal{W}_S \times \mathcal{B}_P \times \mathcal{B}_S$ be the *set of WESA data flow types*.

4.2.4 WESA resources

Definition 4.12 (WESA resource layout)

WESA resource layout defines what kind of resources host the WESA nodes. Based on definition 2.33 and 4.1, mediator nodes have to provide custom services which are not available on computational or storage machines. Therefore, mediator nodes always have to be hosted on dedicated or on external machines similarly to all previous cases. Engine and workflow repository nodes have to provide the executable engines and workflows. Similarly to DRC repositories, computational machines cannot be utilized for this purpose, but this functionality can be provided by the services of storage machines, by custom services hosted on dedicated machines, or by services running on external machines. Engine execution nodes cannot run on storage resources, but they can run on computational, dedicated, or external machines. Having these, the set of all possible WESA resource layouts is defined over node type set $\mathcal{T}$ as:

$$\begin{aligned}\mathcal{RL} := \{\xi \in \mathcal{L}_R(\mathcal{T}) \| \xi_M \in \{DeM, ExM\} \wedge \xi_{RE} \neq CoM \wedge \xi_{RW} \neq CoM \wedge \\ \wedge \xi_E \neq StM \wedge \xi_A, \xi_J = ExM\}\end{aligned} \tag{4.1}$$

4.2.5 WESA interface

Definition 4.13 (WESA frontend interface)

Frontend interface ($\mathcal{I}_F$) is the interface through which applications can utilize the provided functionality of a WESA. Let $\mathcal{I}_F := \{Gen, Spe\}$ be the set of application interface types. (See definition 2.36.)

Representation of $\mathcal{I}_F$ (see definition 2.35) is not considered as part of a WESA architecture, because mappings between representations are straightforward to realise. However, generality is vital, since it determines the set of workflow engine requests

that can be performed via a WESA. Hence,

Definition 4.14 (WESA backend interface)
Backend interface ($\mathcal{I}_B$) defines how engines can be accessed. Let $\mathcal{I}_B := \{CLI, API\}$ be the set of backend interface types.

Since engine interfaces are designed to interact with a particular workflow engine, engine interface is always specific to a particular workflow engine. However, in terms of engine interface, representation is vital, since it determines how an existing WESA can be extended with the support of further engines. Most workflow systems provide either or both API[1] and CLI representations to interact with their workflow engines.

Definition 4.15 (Set of possible WESA interfaces)
The set of possible interfaces can be defined as $\mathcal{IN} := \mathcal{I}_F \times \mathcal{I}_B$.

4.2.6 WESA architecture and solution

Definition 4.16 (Set of possible WESA architectures)
The *set of possible WESA architectures* is constructed as:

$$\begin{aligned}\mathcal{AR} := \{((h,\delta),\xi,(q_e,s_e,q_w,s_w,q_b,s_b),(i_f,i_b)) \in \mathcal{L}_S \times \mathcal{L}_R \times \mathcal{DF} \times \mathcal{IN} \| \\ q_e, q_w, \text{ and } q_b \text{ are acyclic path layouts based on instance layout } h \wedge \quad (i) \\ \wedge \forall (t_1,t_2) \in h : \xi_{t_1} = \xi_{t_2}\}. \quad (ii)\end{aligned} \tag{4.2}$$

Note that conditions are needed for the same reasons as described in the case of DASG architectures in definition 2.40.

[1]This usually means web service interface.

Definition 4.17 (WESA solution)
A WESA solution is a set of WESA architectures. In words, it is a not empty subset of $\mathcal{AR}$.

4.3 WESA architecture analysis

4.3.1 WESA generality and extendibility

Generality of a WESA solution depends on the frontend interface. By applying a specific frontend interface the usage of a solution can be simplified, but this also restricts the provided functionality and the set of workflows that can be executed via a particular WESA solution. In order to enable the execution of the widest possible set of workflows, frontend interfaces should not restrict the data type and number of input and output parameters that can be specified for a workflow.

Extendibility of a WESA solution is determined by how easy it is to extend the set of available workflow engines, which is defined by the backend interface of a WESA solution. CLI backend interface is recommended, since it enables the straightforward extension of the set of supported workflow engines without requiring programming skills. Furthermore, the mediator knows about the available engine repositories and the available engines. If the system is extended with a new workflow engine, the mediator has to be updated. In the case of instance layouts that have (A, M) coupled, each application has a copy of the mediator, in which case each application has to be updated with the new mediator version. However, if the mediator is not coupled with the application and runs as a centralised service, once that service is updated, all applications can use the new workflow engine. Therefore, it is recommended to have a centralised mediator which is not coupled with the application and is hosted on a dedicated machine.

4.3.2 WESA performance

The aim of the performance analysis is to compare overhead, latency, and scalability of different WESA architectures and show how these values vary with bulk data volume, engine size, workflow size, and number of simultaneous requests. The performance comparison is based on WESA scenarios where $r \in \mathbb{N}^+$ different applications hosted by different machines initiate the execution of r different workflows. The first job of each workflow is a job that receives data from the application that initiated the execution. These scenarios are represented as the elements of the set defined below.

Definition 4.18 (WESA scenarios)

$$
\begin{aligned}
\text{Let } \mathcal{AS} := \{((h,\delta),\xi,q_e,q_w,q_b,w_e,s_e,s_w,s_b,l_e,l_w,l_b,r) \in \\
\in \mathcal{L}_S \times \mathcal{L}_R \times \mathcal{E}_P \times \mathcal{W}_P \times \mathcal{B}_P \times \mathbb{R}_0^+ \times (\mathbb{N}^+)^7 \| \\
\delta_A = r \wedge & \quad (i) \\
\wedge q_e, q_w \text{ and } q_b \text{ are acyclic path layouts based on } h \wedge & \quad (ii) \\
\wedge \forall (t_1,t_2) \in h : \xi_{t_1} = \xi_{t_2} \wedge & \quad (iii) \\
\wedge \xi_E \neq CoM \Rightarrow w_e = 0 \wedge & \quad (iv) \\
\wedge \forall t \in \mathcal{T} \backslash \{A\} : r \equiv 0 \mod \delta_t \wedge & \quad (v) \\
\wedge l_e \equiv 0 \mod s_e \wedge l_w \equiv 0 \mod s_w \wedge l_b \equiv 0 \mod s_b\}. & \quad (vi)
\end{aligned}
\tag{4.3}
$$

be the *set of analytical WESA scenarios*. Let $a = ((h,\delta),\xi,q_e,q_w,q_b,w_e,s_e,s_w,s_b,l_e,l_w,l_b,r) \in \mathcal{AS}$ be an analytical scenario. Condition (i) ensures that none of the application nodes are coupled, all are hosted on different machines. a determines structure layout, resource layout, and data flow of a DASG architecture, this is ensured by conditions (ii) and (iii).

By (h,δ) and ξ a explicitly defines a WESA structure and a resource layout. In

terms of *data flow*, a defines engine, workflow and bulk data path layout explicitly by q_e, q_w and q_b. Engine, workflow and bulk data size are represented by l_e, l_w and l_b, Engine, workflow and bulk data slice size by s_e, s_w and s_b. These implicitly define engine, workflow and bulk data staging as $k_e := \frac{l_e}{s_e}$, $k_w := \frac{l_w}{s_w}$ and $k_b := \frac{l_b}{s_b}$, where staging is non pipelined if slice number equals to 1 and pipelined otherwise. Note that condition (vi) ensures that l_e is dividable by s_e, l_w is dividable by s_w, and l_b is dividable by s_b. w_e represents the delay resulted by the engine waiting in a job queue before it is scheduled for execution when it is executed on a computational machine. This number is constant and it also can be 0 representing cases where job queues are empty or there are no queues at all. Condition (iv) ensures that w_e is always 0 if the engine is not executed on a computational machine.

Furthermore, condition (v) ensures that all nodes of a given node type other than A are coupled with equal number of nodes of that particular type. This ensures that nodes of each node type can be equally distributed between the available machines.

Definition 4.19 (WESA scenario execution)
Since control flow is excluded from the model, the analysis is based on engine, workflow, and bulk data flow. $\forall i \in [1..r]$: let $e_i \in \mathcal{B}$ byte array represent the engine that executes workflow w_i (see below) and to be transferred from RE_i to E_i, $w_i \in \mathcal{B}$ byte array represent the workflow that is to be invoked by application A_i and to be transferred from RW_i to E_i, and $b_i \in \mathcal{B}$ byte array represent the bulk data that is to be transferred from A_i to J_i. A WESA scenario is executed in four steps:

1. engine transfer: $\forall i \in [1..r]$: e_i is transferred from RE_i to E_i via path $\psi_i(q_e)$ simultaneously,

2. workflow transfer: $\forall i \in [1..r]$: w_i is transferred from RW_i to E_i via path $\psi_i(q_w)$ simultaneously,

3. engine queuing: all engines are waiting w_e amount of time to be scheduled for execution,

4. bulk data transfer: $\forall i \in [1..r]$: the execution of the ith workflow engine starts and b_i is transferred from A_i to J_i via path $\psi_i(q_b)$ simultaneously.

Engine flow can be represented as a simultaneous transfer (see definition 2.48), since conditions of definition 4.18 ensure that $((h, \delta), q_e, s_e, l_e, r) \in \mathcal{D}_{st}$. Similarly, workflow and bulk data flow also can be represented as: $((h, \delta), q_w, s_w, l_w, r)$, $((h, \delta), q_b, s_b, l_b, r)$ $\in \mathcal{D}_{st}$. Having these, the performance functions of a scenario can be defined as follows.

Definition 4.20 (Performance of WESA engine and workflow transfer) Let $a := ((h, \delta), \xi, q_e, q_w, q_b, w_e, s_e, s_w, s_b, l_e, l_w, l_b, r) \in \mathcal{AS}$, and Γ_e, $\Gamma_w : \mathcal{AS} \to \mathbb{R}_0^+$ be functions for determining respectively engine and workflow transfer time, where:

$$\Gamma_e(a) := \tau_q((h, \delta), q_e, s_e, l_e, r) \qquad \Gamma_w(a) := \tau_q((h, \delta), q_w, s_w, l_w, r). \tag{4.4}$$

Note that definition 2.49 is applied to identify $\Gamma_e(a)$ and $\Gamma_w(a)$.

Definition 4.21 (Performance of WESA bulk data transfer) Let $a := ((h, \delta), \xi, q_e, q_w, q_b, w_e, s_e, s_w, s_b, l_e, l_w, l_b, r) \in \mathcal{AS}$, and $\Gamma_b, \Delta_b, \Theta_b : \mathcal{AS} \to \mathbb{R}_0^+$ be functions for determining respectively bulk data transfer time, overhead, and latency as:

$$\Gamma_b(a) := \tau_q((h, \delta), q_b, s_b, l_b, r), \tag{4.5}$$

$$\Delta_b(a) := \Gamma_b(a) - \chi((A, J) \notin h) k_b s_b K, \text{ and} \tag{4.6}$$

$$\Theta_b(a) := \epsilon_q((h, \delta), q_b, s_b, l_b, r). \tag{4.7}$$

Note that definition 2.49 is applied to determine $\Gamma_b(a)$ and $\Theta_b(a)$. $\forall i \in [1..r]$: transferring b_i directly between A_i and J_i takes $\tau_e(\lambda(b_i), (A_i, J_i))$ time. This value is

0 if $(A, J) \in h$ and $k_b s_b K$ otherwise. Overhead on bulk data transfer of a particular scenario is considered as this time subtracted from bulk data transfer time.

Definition 4.22 (Overall WESA performance functions)
Let $\Gamma, \Delta, \Theta : \mathcal{AS} \rightarrow \mathbb{R}_0^+$ be functions for determining respectively execution time, overhead, and latency of a scenario, where:

$$\Gamma(a) := w_d + \Gamma_e(a) + \Gamma_w(a) + \Gamma_b(a), \tag{4.8}$$

$$\Delta(a) := w_d + \Gamma_e(a) + \Gamma_w(a) + \Delta_b(a), \text{ and} \tag{4.9}$$

$$\Theta(a) := w_d + \Gamma_e(a) + \Gamma_w(a) + \Theta_b(a). \tag{4.10}$$

Because engine and workflow transfer always has to be performed before workflow execution, engine and workflow transfer times are always added to the overall latency and overhead of a scenario.

Definition 4.23 (Scalability of WESA data transfer)
Performance functions of any WESW scenario are characterised based on growth rates in function of l_w, l_e, l_b, and r. It is represented by the Bachmann–Landau (Big O) notation in analogy with all previous contributions.

4.3.3 WESA bulk data flow

Based definition 2.54 and the four bulk data path layout types ((A, J), (A, M, J), (A, E, J), (A, M, E, J)) defined in definition 4.10, 18 different bulk data flow cases can be identified. These are listed in table 4.3. Restrictions on type layout implied by instance layout are also illustrated in the table. Based on definition 2.55, bulk data flow cases can be divided into 5 different groups, where cases of each group are equivalent, in terms that they have the same performance characteristics. These groups and representative (which are marked with asterisks) bulk data flow cases (see definition 2.54) are also illustrated in the table 4.3.

Case	Bulk data path	Instance layout	Restrictions	Group
BC1 *	(A, J)	$(A, J) \in h$	$\delta_J = r$	BG1
BC2	(A, M, J)	$(A, M), (M, J) \in h$	$\delta_M, \delta_J = r$	
BC3	(A, E, J)	$(A, E), (E, J) \in h$	$\delta_J, \delta_E = r$	
BC4	(A, M, E, J)	$(A, M), (M, E), (E, J) \in h$	$\delta_J, \delta_M, \delta_E = r$	
BC5 *	(A, J)	$(A, J) \notin h$		BG2
BC6	(A, M, J)	$(A, M) \in h \wedge (M, J) \notin h$	$\delta_M = r$	
BC7	(A, M, J)	$(A, M) \notin h \wedge (M, J) \in h$	$\delta_M = \delta_J$	
BC8	(A, E, J)	$(A, E) \in h \wedge (E, J) \notin h$	$\delta_E = r$	
BC9	(A, E, J)	$(A, E) \notin h \wedge (E, J) \in h$	$\delta_E = \delta_J$	
BC10	(A, M, E, J)	$(A, M) \notin h \wedge (M, E), (E, J) \in h$	$\delta_J = \delta_M = \delta_E$	
BC11	(A, M, E, J)	$(M, E) \notin h \wedge (E, J), (A, M) \in h$	$\delta_M = r$ $\delta_J = \delta_E$	
BC12	(A, M, E, J)	$(E, J) \notin h \wedge (M, E), (A, M) \in h$	$\delta_M, \delta_E = r$	
BC13 *	(A, M, J)	$(A, M), (M, J) \notin h$		BG3
BC14	(A, M, E, J)	$(M, E), (A, M) \notin h \wedge (E, J) \in h$	$\delta_J = \delta_E$	
BC15	(A, M, E, J)	$(E, J), (A, M) \notin h \wedge (M, E) \in h$	$\delta_M = \delta_E$	BG3 $\wedge$ BG4
BC16 *	(A, E, J)	$(A, E), (E, J) \notin h$		BG4
BC17	(A, M, E, J)	$(M, E), (E, J) \notin h \wedge (A, M) \in h$	$\delta_M = r$	
BC18 *	(A, M, E, J)	$(A, M), (E, J), (M, E) \notin h$		BG5

Table 4.3: WESA bulk data flow cases

Group	Transfer time ($\Gamma_b(a)$)
BG1	0
BG2	$\frac{rk_bs_bK}{\min\{r,\delta_J\}}$
BG3	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rk_bs_bK}{\min\{r,\delta_M,\delta_J\}}$
BG4	$\frac{rs_bK}{\min\{r,\delta_E\}} + \frac{rk_bs_bK}{\min\{r,\delta_E,\delta_J\}}$
BG5	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rs_bK}{\min\{r,\delta_M,\delta_E\}} + \frac{rs_bK}{\min\{r,\delta_E,\delta_J\}} + \frac{r(k_b-1)s_bK}{\min\{r,\delta_M,\delta_E,\delta_J\}}$

Table 4.4: Time of WESA bulk data transfer

Transfer time, overhead, and latency values can be found in table 4.4, 4.5, and 4.6 respectively, where the formulas are based on the definition of $\Gamma_b(a)$, $\Delta_b(a)$, and $\Theta_b(a)$. Based on $\Delta_b(a)$ and $\Theta_b(a)$, the architectural conditions which determine scalability in terms of overhead and latency are identified for each group, these can be found in table 4.5, and 4.6. In particular, cases of group BG1 always provide 0 overhead and latency on bulk data staging, cases of group BG2 always provide 0

Group	Overhead ($\Delta_b(a)$)	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
BG1	0	$\nexists$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG2	$\frac{rk_bs_bK}{\min\{r,\delta_J\}} - k_bs_bK$	$\delta_J \geqq r$	$\nexists$	$\nexists$	$\nexists$	$\delta_J < r$
BG3	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rk_bs_bK}{\min\{r,\delta_M,\delta_J\}} - -k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_J \geqq r$ $\delta_M \geqq r$	$k_b = 1$ $\delta_J \geqq r$ $\delta_M \geqq r$	$\nexists$	$\delta_J < r \vee \delta_M < r$
BG4	$\frac{rs_bK}{\min\{r,\delta_E\}} + \frac{rk_bs_bK}{\min\{r,\delta_E,\delta_J\}} - -k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_J \geqq r$ $\delta_E \geqq r$	$k_b = 1$ $\delta_J \geqq r$ $\delta_E \geqq r$	$\nexists$	$\delta_J < r \vee \delta_E < r$
BG5	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rs_bK}{\min\{r,\delta_M,\delta_E\}} + +\frac{rs_bK}{\min\{r,\delta_E,\delta_J\}} + +\frac{r(k_b-1)s_bK}{\min\{r,\delta_M,\delta_E,\delta_J\}} - k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_E \geqq r$ $\delta_M \geqq r$ $\delta_J \geqq r$	$k_b = 1$ $\delta_E \geqq r$ $\delta_M \geqq r$ $\delta_J \geqq r$	$\nexists$	$\delta_E < r \vee \delta_M < r \vee \delta_J < r$

Table 4.5: Overhead and scalability of WESA bulk data staging

Group	Latency ($\Theta_b(a)$)	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
BG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG2	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG3	$\frac{rs_bK}{\min\{r,\delta_M\}}$	$\nexists$	$k_b > 1$ $\delta_M \geqq r$	$k_b = 1$ $\delta_M \geqq r$	$k_b > 1$ $\delta_M < r$	$k_b = 1$ $\delta_M < r$
BG4	$\frac{rs_bK}{\min\{r,\delta_E\}}$	$\nexists$	$k_b > 1$ $\delta_E \geqq r$	$k_b = 1$ $\delta_E \geqq r$	$k_b > 1$ $\delta_E < r$	$k_b = 1$ $\delta_E < r$
BG5	$\frac{rs_bK}{\min\{r,\delta_M\}} + +\frac{rs_bK}{\min\{r,\delta_M,\delta_E\}}$	$\nexists$	$k_b > 1$ $\delta_E \geqq r$ $\delta_M \geqq r$	$k_b = 1$ $\delta_E \geqq r$ $\delta_M \geqq r$	$k_b > 1$ $\delta_E < r \vee \delta_M < r$	$k_b = 1$ $\delta_E < r \vee \delta_M < r$

Table 4.6: Latency and scalability of WESA bulk data staging

latency, but overhead is 0 if and only if $\delta_J \geqq r$, otherwise overhead is linear with both l_b and r. Cases of group BG2, BG3, BG4, and BG5 never provide 0 latency nor overhead and the same rules apply as in the case of DASG bulk data transfer groups BG2 and BG3 described in section 2.3.3.

4.3.4 WESA engine and workflow flow

Based definition 2.54 the two engine path layout types ((RE, E), (RE, M, E)) and the two workflow path layout types ($(RW, E), (RW, M, E)$), 6 different engine data flow cases and 6 different workflow data flow case can be identified. These are listed in table 4.7 and 4.9. Representative cases (see definition 2.54) are marked with asterisks. Restrictions implied by instance layout are also illustrated in the table.

Case	Engine path	Instance layout	Restrictions	Group
EC1 *	(RE, E)	$(RE, E) \in h$	$\delta_{RE} = \delta_E$	EG1
EC2	(RE, M, E)	$(RE, M), (M, E) \in h$	$\delta_M = \delta_{RE} = \delta_E$	
EC3 *	(RE, E)	$(RE, E) \notin h$		EG2
EC4	(RE, M, E)	$(RE, M) \in h \wedge (M, E) \notin h$	$\delta_M = \delta_{RE}$	
EC5	(RE, M, E)	$(RE, M) \notin h \wedge (M, E) \in h$	$\delta_E = \delta_M$	
EC6 *	(RE, M, E)	$(RE, M), (M, E) \notin h$		EG3

Table 4.7: WESA engine data flow cases

Group	Transfer time ($\Gamma_e(a)$)	$O(0)$	$O(1)$	$O(l_e)$	$O(r)$	$O(rl_e)$
EG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
EG2	$\frac{rk_e s_e K}{\min\{r, \delta_E, \delta_{RE}\}}$	$\nexists$	$\nexists$	$\delta_{RE} \geqq r$ $\delta_E \geqq r$	$\nexists$	$\delta_{RE} < r \vee \delta_E < r$
EG3	$\frac{rs_e K}{\min\{r, \delta_{RE}, \delta_M\}} +$ $+ \frac{rs_e K}{\min\{r, \delta_M, \delta_E\}} +$ $+ \frac{r(k_e - 1)s_d K}{\min\{r, \delta_M, \delta_{RE}, \delta_E\}}$	$\nexists$	$\nexists$	$\delta_M \geqq r$ $\delta_E \geqq r$ $\delta_{RE} \geqq r$	$\nexists$	$\delta_M < r \vee \delta_{RE} < r \vee \delta_E < r$

Table 4.8: Transfer time and scalability of WESA engine transfer

In the case of both engine and workflow transfer, the 6 cases can be divided into 3 groups. Performance properties are the same within the each group. Transfer time values are determined based on the definition of $\Gamma_e(a)$, $\Gamma_w(a)$ and included in table 4.8 and 4.10.

Cases of group EG1 and WG1 require, that all nodes through which the engines/workflows are transferred are hosted on the same machine, implying that transfer time is 0. In cases of group EG2, EG3, WG2, and WG3, the following rules

Case	Workflow path	Instance layout	Restrictions	Group
WC1 *	(RW, E)	$(RW, E) \in h$	$\delta_{RW} = \delta_E$	WG1
WC2	(RW, M, E)	$(RW, M), (M, E) \in h$	$\delta_M = \delta_{RW} = \delta_E$	
WC3 *	(RW, E)	$(RW, E) \notin h$		WG2
WC4	(RW, M, E)	$(RW, M) \in h \wedge (M, E) \notin h$	$\delta_M = \delta_{RW}$	
WC5	(RW, M, E)	$(RW, M) \notin h \wedge (M, E) \in h$	$\delta_E = \delta_M$	
WC6 *	(RW, M, E)	$(RW, M), (M, E) \notin h$		WG3

Table 4.9: WESA workflow data flow cases

Group	Transfer time $\Gamma_w(a))$	$O(0)$	$O(1)$	$O(l_w)$	$O(r)$	$O(rl_w)$
WG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
WG2	$\frac{rk_w s_w K}{\min\{r, \delta_E, \delta_{RW}\}}$	$\nexists$	$\nexists$	$\delta_{RW} \geqq r$ $\delta_E \geqq r$	$\nexists$	$\delta_{RW} < r \vee \delta_E < r$
WG3	$\frac{rs_w K}{\min\{r, \delta_{RW}, \delta_M\}} +$ $+ \frac{rs_w K}{\min\{r, \delta_M, \delta_E\}} +$ $+ \frac{r(k_w - 1)s_w K}{\min\{r, \delta_M, \delta_{RW}, \delta_E\}}$	$\nexists$	$\nexists$	$\delta_M \geqq r$ $\delta_E \geqq r$ $\delta_{RW} \geqq r$	$\nexists$	$\delta_M < r \vee \delta_{RW} < r \vee \delta_E < r$

Table 4.10: Transfer time and scalability of WESA workflow transfer

apply. As long as r is not greater than any of the δ values in the transfer time formula of a particular group, transfer time is independent of r. In this case scalability is $O(l_e)$ in the case of engine transfer and $O(l_w)$ in the case of workflow transfer. If r is greater than any of the δ values in the transfer time formula of a particular group, than transfer time is linear with r. In this case scalability is $O(rl_e)$ in the case of engine transfer and $O(rl_w)$ in the case of workflow transfer. Scalability values are also shown in table 4.8 and 4.10.

4.3.5 Recommended WESA structure layout, data flow, and resource layout

In any of the 18 bulk data transfer cases, it is possible to realise scalable bulk data transfer where both overhead and latency on bulk data transfer are independent of

r and l_b. In terms of type layout, while cases BC1 and BC5 only require that $\delta_J \geqq r$, cases BC2, BC6, BC7, and BC13 require that $\delta_M, \delta_J \geqq r$, cases BC3, BC8, BC9, BC16 require that $\delta_E, \delta_J \geqq r$, while cases BC4, BC10, BC11, BC12, BC14, BC15, BC17, and BC18 require that all $\delta_J, \delta_M, \delta_E \geqq r$. In terms of bulk data staging BC13 - BC18 require pipelined staging ($k_b > 1$) as well.

In all of the 6 engine and 6 workflow transfer cases, transfer time is independent of l_b and in any of the cases it is possible to realise scalable transfer where engine/workflow transfer time is independent of r. EC1 and EC2 always provide scalable engine, WC1 and WC2 always provide scalable workflow transfer where transfer time is independent of r. In terms of type layout, EC3 provides scalable engine transfer if $\delta_E \geqq r$ and $\delta_{RE} \geqq r$, WC3 provides scalable workflow transfer if $\delta_E \geqq r$ and $\delta_{RW} \geqq r$, while EC4, EC5, EC6, WC4, WC5, and WC6 also require that $\delta_M \geqq r$. The selection of proposed structure layout, resource layout and data flow combinations is based on the following recommendations:

R1 - Mediator Similarly to DASGs and DASWs and based section 4.3.1, the mediator should be established as a centralised service hosted on a dedicated machine ($\xi_M = DeM$). Based on this and on definition 2.32, the larger the number of utilised dedicated machines is, the more cost demanding it is to set up and maintain a WESA. For this reason, it is amied to minimise δ_M (let $\delta_M = 1$). This has several implications. Data flow cases that require that $\delta_M \geqq r$ are not recommended. In each scenario $\delta_A = r$ is always true. Therefore, data flow cases which require to have (A, M) coupled are not recommended. Moreover, since bulk data amount is multiple orders of magnitude greater than engine or workflow size, it is always aimed to minimise overhead and latency on bulk data transfer and make it independent of the number of simultaneous requests and bulk data amount. This means that data flow cases which are transferring bulk data via the mediator are not recommended

Recommendation	WESA data flow case
R1	BC2, BC4, BC6, BC7, BC10-BC15, BC17, BC18, EC2, EC5, WC2, WC5
R2	BC1-BC4, BC8
R3	EC5, EC6, WC5, WC6

Table 4.11: Elimination of WESA data flow cases based on different recommendations.

and $\delta_J \geqq r$ should always be provided. This also means that cases that require $(M, J) \in h$ are not recommended. Furthermore, although it is not included in the model, in order to distribute the computational load on the mediator machine, instance layouts having (M, E) coupled are not recommended either.

R2 - Repositories and engine execution In terms of instance layout, cases that require $(A, E) \in h$ or $(A, J) \in h$ cannot be recommended, since it cannot be guaranteed in general that software and network requirements of jobs/workflow engines can be fulfilled. This means that $\xi_E \neq ExM$. If engines are executed on computational machines ($\xi_E = CoM$), then cases that require (RW, E) or (RE, E) coupled are not recommended, since according to definition 4.12: $\xi_{RE}, \xi_{RW} \neq CoM$. However, if engines are executed on dedicated machines ($\xi_E = DeM$), then cases that require (RW, E) or (RE, E) coupled can be recommended, since although the number of dedicated machines should be minimised, in special cases $\xi_E = DeM$ can be recommended if this way better performance can be achieved. (See case 1/b-i in the followings.)

R3 - Engine and workflow transfer In cases where $(RE, M) \notin h$, engines should not be transferred via the mediator, because this increases overhead and if $\delta_{RE} > 1$ the mediator also bottlenecks data transfer in case of multiple simultaneous requests. The same is true for workflows if $(RW, M) \notin h$. Therefore, if $(RE, M) \notin h$, then only engine path layout $\{RE, E\}$ is recommended, and if $(RW, M) \notin h$, then

only workflow path layout $\{RW, E\}$ is recommended.

Aspects			Proposed 1 (PC1)	Proposed 2 (PC2)	Proposed 3 (PC3)	Proposed 4 (PC4)	Proposed 5 (PC5)	Proposed 6 (PC6)	Proposed 7 (PC7)	Proposed 8 (PC8a,PC8b)	Proposed 9 (PC9a,PC9b)	Proposed 10 (PC10a,PC10b)	Proposed 11 (PC11a,PC11b)	Proposed 12 (PC12a,PC12b)	Proposed 13 (PC13a,PC13b)	Proposed 14 (PC14a,PC14b)	Proposed 15 (PC15)	Proposed 16 (PC16)	Proposed 17 (PC17)	Proposed 18 (PC18)
Structure	Instance layout	(A,M)																		
		(A,RE)																		
		(A,RW)					X	X						X	X				X	
		(A,E)																		
		(M,RE)		X				X	X		X				X	X				
		(M,RW)			X				X			X				X		X		
		(M,E)																		
		(RW,RE)				X			X				X			X				X
		(RW,E)																		X
		(RE,E)															X	X	X	X
	Type layout	δ_M	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		δ_{RE}	≥1	1	≥1	≥1	≥1	1	1	≥1	1	≥1	≥1	≥1	1	1	≥1	≥1	≥1	≥1
		δ_{RW}	≥1	≥1	1	≥1	≥r	≥r	1	≥1	≥1	1	≥1	≥r	≥r	1	≥1	1	r	≥1
		δ_E	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥1	≥1	≥1	≥1
Data flow	Bulk data path	(A,J)	O	O	O	O	O	O	O								X	X	X	X
		(A,M,J)																		
		(A,E,J)	O*	O*	O*	O*	O*	O*	O*	X	X	X	X	X	X	X				
		(A,M,E,J)																		
	Bulk data staging	Pipelined (k_s=1)	O	O	O	O	O	O	O	X^a	X^a	X^a	X^a	X^a	X^a	X^a	O	O	O	O
		Not pipelined (k_s>1)	O	O	O	O	O	O	O	X^b	X^b	X^b	X^b	X^b	X^b	X^b	O	O	O	O
	Engine path	(RE,E)	X	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		(RE,M,E)		O				O	O		O				O	O				
	Engine staging	Pipelined (k_g>1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_g=1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
	Workflow path	(RW,E)	X	X	O	X	X	X	O	X	X	O	X	X	X	O	X	O	X	X
		(RW,M,E)			O				O			O				O		O		
	Workflow staging	Pipelined (k_g>1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_g=1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Resource	Resource layout	Mediator	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM
		Engine Repository	StM	DeM	StM	StM	StM	DeM	DeM	StM	DeM	StM	StM	StM	DeM	DeM	DeM	DeM	DeM	DeM
		Workflow Repository	StM	StM	DeM	StM	ExM	ExM	DeM	StM	StM	DeM	StM	ExM	ExM	DeM	StM	DeM	ExM	DeM
		Engine execution	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	DeM	DeM	DeM	DeM

Table 4.12: Proposed WESA structures, data flows, and resource layouts, where sign X within instance layout shows which nodes are coupled in a particular case. Sign X within data flow means that only the correlated architectural approach can be implemented, sign O indicates that it is irrelevant which approach is chosen, while sign O* means the same as O, but it only applies if instance layout defines node types E and J are coupled. Sign X^a and X^b mean that it is possible to realise both pipelined (X^a) and non pipelined (X^b) bulk data staging by the given structure, but this affects the performance characteristics as illustrated in table 4.14.

Based on these recommendations, data flow cases listed in table 4.11 are excluded. Hence, data transfer cases BC5, BC9, BC16, engine transfer cases EC1, EC3, EC4, and workflow transfer cases WC1, WC3, WC4 can be recommended. Table 4.12 illustrates all combinations of these data flow cases and structure layouts that can implement them. Different data flow types can be recommended under

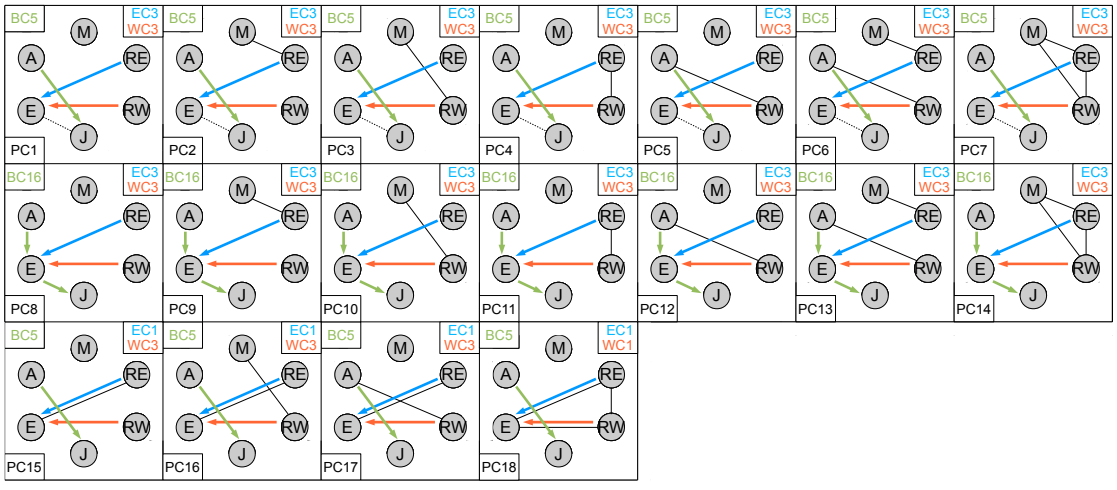

Figure 4.3: Combinations of recommended WESA data flow cases, where black lines represent which node types are coupled, dashed lines represent optional couplings, green arrows represent bulk data path layouts, blue arrows represent engine path layouts, and orange arrows represent workflow path layouts.

different circumstances. These are classified based on the following 3 aspects:

1. Workflow jobs

- **(a)** If workflow jobs are executed locally to the workflow engine ($(E, J) \in h$), then only BC5 or BC9 can be recommended and $\xi_E = CoM$, BC16 cannot be applied, since it explicitly defines that $(E, J) \notin h$. In this case it is not recommended to use workflow engines installed on dedicated machines, since this may bottleneck bulk data transfer in the case of large number of simultaneous requests. Hence, in this case EC1 is not recommended.

- **(b)** If workflow jobs are executed remotely to the workflow engine ($(E, J) \notin h$), then both BC5 and BC16 can be recommended.

  (b-i) If this is the case and bulk data can be transferred directly (i.e using GridFTP or other data transfer protocol) to/from the job then it is rec-

ommended to use previously installed workflow engines running on dedicated machines, since this way engine transfer does not increase overhead and latency and the engine will not bottleneck bulk data transfer either. In this case BC5 combined with EC1 is recommended and $\xi_E = DeM$.

(b-ii) If data can be transferred to the job only via the workflow engine then only BC16 can be applied and $\xi_E = CoM$. For the same reason as described in the case of locally executed jobs (see case 1/a) EC1 is not recommended.

2. Workflow engines

- **(a)** If workflow engines are relatively small (up to a few megabytes), then $(M, RE) \in h$ can be recommended which means that both EC3 and EC4 can be applied and $\xi_{RE} = DeM$, but
- **(b)** if they are relatively large (hundreds of megabytes), then $(M, RE) \notin h$ is recommended with $\xi_{RE} = StM$, since it allows to utilize multiple distributed engine repositories to distribute the load on engine transfer. In this case only EC3 can be applied.

However, in special cases (see case 1/b-i) it is recommended to use workflow engines hosted on dedicated machines where EC1 is recommended independently of engine size. In this case $\xi_{RE} = DeM$.

3. Workflow descriptors

- **(a)** If workflows are not provided by the application ($(A, RW) \notin h$) and

  (a-i) workflow descriptors are relatively small (up to a few megabytes), then the workflow repository can be coupled with the mediator ($(M, RW) \in h$),

in which case both WC3 and WC4 can be recommended where $\xi_{RW} = DeM$. However, if

(a-ii) workflow descriptors are relatively large (hundreds of megabytes), then $(M, RW) \notin h$ should be applied, since, like in the case of engines, this allows to utilize multiple storage machines as workflow repositories to distribute the load on workflow transfer. In this case the workflow repository can be hosted on a storage machine ($\xi_{RW} = StM$) or in the special case where the engine is hosted on a dedicated machine (see case 1/b-i), then WC1 can be recommended in which case $\xi_{RW} = DeM$ and $(E, RW) \in h$.

- **(b)** Cases where workflows are provided by the application are represented by $(A, RW) \in h$. In this case $\xi_{RW} = ExM$. In the case of the latter two (a-ii and b) workflow transfer path type WC3 can be recommended, since WC5 transfers the workflows via the mediator which may bottleneck the transfer.

However, in special cases where engine repository and execution are coupled and hosted on the same DeM, then RW nodes can also be hosted on a these machines $((RE, E), (RW, E) \in h, \xi_{RW} = ReM)$ to avoid latency and overhead on workflow transfer. In this case WC1 is recommended.

Case	Proposed combinations
1/a, 2/a, 3/a-i	PC7
1/a, 2/a, 3/a-ii	PC2
1/a, 2/a, 3/b	PC6
1/a, 2/b, 3/a-i	PC3
1/a, 2/b, 3/a-ii	PC1, PC4
1/a, 2/b, 3/b	PC5
1/b-i, 2/a, 3/a-i	PC16, PC18
1/b-i, 2/a, 3/a-ii	PC15, PC18
1/b-i, 2/a, 3/b	PC17

Case	Proposed combinations
1/b-i, 2/b, 3/a-i	PC16, PC18
1/b-i, 2/b, 3/a-ii	PC15, PC18
1/b-i, 2/b, 3/b	PC17
1/b-ii, 2/a, 3/a-i	PC14
1/b-ii, 2/a, 3/a-ii	PC9
1/b-ii, 2/a, 3/b	PC13
1/b-ii, 2/b, 3/a-i	PC10
1/b-ii, 2/b, 3/a-ii	PC8, PC11
1/b-ii, 2/b, 3/b	PC12

Table 4.13: Proposed WESA structure and data flow combinations in different cases

See table 4.13, that summarizes which proposed structure and data flow combi-

nations are recommended in the different cases. Table 4.14 illustrates performance characteristics of each proposed combination. Based on definition 2.54, BC5 is the representative of the group that also includes BC9. Note that, since BC5 and BC9 are equivalent (see definition 2.55), their performance characteristics are the same. This also applies for EC3 and EC4; and WC3 and WC4. Combinations of representative cases of are shown in figure 4.3.

Proposed	Overhead / Latency	Overhead / Latency scalability
PC1, PC4	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + \frac{rl_wK}{\min\{r,\delta_{RW}\}}$	$O(rl_e + rl_w + 1)$
PC2	$w_e + rl_eK + \frac{rl_wK}{\min\{r,\delta_{RW}\}}$	$O(rl_e + rl_w + 1)$
PC3	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + rl_wK$	$O(rl_e + rl_w + 1)$
PC5	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + l_wK$	$O(rl_e + l_w + 1)$
PC6	$w_e + rl_eK + l_wK$	$O(rl_e + l_w + 1)$
PC7	$w_e + rl_eK + rl_wK$	$O(rl_e + rl_w + 1)$
PC8a, PC11a	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + s_bK$	$O(rl_e + rl_w + 1)$
PC8b, PC11b	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC9a	$w_e + rl_eK + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + s_bK$	$O(rl_e + rl_w + 1)$
PC9b	$w_e + rl_eK + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC10a	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + rl_wK + s_bK$	$O(rl_e + rl_w + 1)$
PC10b	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + rl_wK + l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC12a	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + l_wK + s_bK$	$O(rl_e + l_w + 1)$
PC12b	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + l_wK + l_bK$	$O(rl_e + l_w + l_b + 1)$
PC13a	$w_e + rl_eK + l_wK + s_bK$	$O(rl_e + l_w + 1)$
PC13b	$w_e + rl_eK + l_wK + l_bK$	$O(rl_e + l_w + l_b + 1)$
PC14a	$w_e + rl_eK + rl_wK + s_bK$	$O(rl_e + rl_w + 1)$
PC14b	$w_e + rl_eK + rl_wK + l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC15	$\frac{rl_wK}{\min\{r,\delta_{RW}\}}$	$O(rl_w)$
PC16	rl_wK	$O(rl_w)$
PC17	l_wK	$O(l_w)$
PC18	0	$O(0)$

Table 4.14: Performance characteristics of proposed WESA structures and data flows, where cases marked with a and b are representing pipelined (a) and non pipelined (b) bulk data staging. Bulk data staging does not affect the performance properties of cases which are not marked.

4.4 Existing and proposed WESA solutions

4.4.1 Existing WESA solution

Although, there are no general coarse grained solutions for heterogeneous workflow engine execution, a solution for runtime workflow interoperability was developed within the SIMDAT project [111, 73, 112, 113]. This solution is based on the Gria service [114] and was designed to enable a few particular workflows of different kinds to invoke each other, but theoretically this approach also can be used as a general solution for invoking heterogeneous workflow engines. The solution wraps the functionality of different workflow engines and makes them accessible via a Web/Grid service based general frontend. This approach uses a backend API for wrapping the engines.

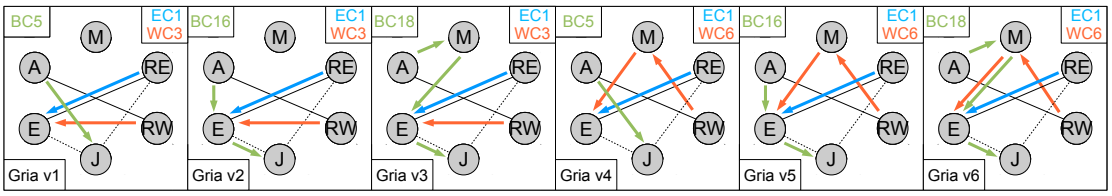

Figure 4.4: Combinations of possible Gria based structures and data flow cases, where color coded arrows and lines represent the same concepts as in the case of figure 4.3.

The Gria Service provides access to previously installed workflow engines hosted by dedicated machines. Hence, there is no engine repository, engines are executed where they reside, meaning that $(RE, E) \in h$. Workflows are provided by the application that invokes the service, workflow repository is not provided. This means that $(A, RW) \in h$. The application, the mediator, and engine are not coupled. Based on

Aspects			Existing						Proposed																
		Architectures	Existing 1 (GRIA v1)	Existing 2 (GRIA v2a,v2b)	Existing 3 (GRIA v3)	Existing 4 (GRIA v4)	Existing 5 (GRIA v5a,v5b)	Existing 6 (GRIA v6)	Proposed 1 (PC1)	Proposed 2 (PC2)	Proposed 3 (PC3)	Proposed 4 (PC4)	Proposed 5 (PC5)	Proposed 6 (PC6)	Proposed 7 (PC7)	Proposed 8 (PC8a,PC8b)	Proposed 9 (PC9a,PC9b)	Proposed 10 (PC10a,PC10b)	Proposed 11 (PC11a,PC11b)	Proposed 12 (PC12a,PC12b)	Proposed 13 (PC13a,PC13b)	Proposed 14 (PC14a,PC14b)	Proposed 15 (PC15)	Proposed 16 (PC16)	Proposed 17 (PC17)	Proposed 18 (PC18)
Structure	Instance layout	(A,M)																								
		(A,RE)																								
		(A,RW)	X	X	X	X	X	X					X	X						X	X				X	
		(A,E)																								
		(M,RE)								X				X	X		X					X	X			
		(M,RW)									X				X			X				X		X		
		(M,E)																								
		(RW,RE)										X			X				X			X				X
		(RW,E)																								X
		(RE,E)	X	X	X	X	X	X															X	X	X	X
	Type layout	δ_M	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		δ_{RE}	≥1	≥1	≥1	≥1	≥1	≥1	≥1	1	≥1	≥1	≥1	1	1	≥1	1	≥1	≥1	≥1	1	1	≥1	≥1	≥1	≥1
		δ_{RW}	r	r	r	r	r	r	≥1	≥1	1	≥1	≥r	≥r	1	≥1	≥1	1	≥1	≥r	≥r	1	≥1	1	r	≥1
		δ_E	≥1	≥1	≥1	≥1	≥1	≥1	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥r	≥1	≥1	≥1	≥1
Data flow	Bulk data path	(A,J)	X			X			O	O	O	O	O	O	O								X	X	X	X
		(A,M,J)																								
		(A,E,J)	O*	X		O*	X		O*	O*	O*	O*	O*	O*	O*	X	X	X	X	X	X	X				
		(A,M,E,J)			X			X																		
	Bulk data staging	Pipelined (k_b=1)	O	X[a]		O	X[a]		O	O	O	O	O	O	O	X[a]	X[a]	X[a]	X[a]	X[a]	X[a]	X[a]	O	O	O	O
		Not pipelined (k_b>1)	O	X[b]	X	O	X[b]	X	O	O	O	O	O	O	O	X[b]	X[b]	X[b]	X[b]	X[b]	X[b]	X[b]	O	O	O	O
	Engine path	(RE,E)	X	X	X	X	X	X	X	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		(RE,M,E)								O				O	O		O					O	O			
	Engine staging	Pipelined (k_g>1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_g=1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
	Workflow path	(RW,E)	X	X	X				X	X	O	X	X	X	O	X	X	O	X	X	X	O	X	O	X	X
		(RW,M,E)				X	X	X			O				O			O				O		O		
	Workflow staging	Pipelined (k_g>1)	O	O	O				O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_g=1)	O	O	O	X	X	X	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Resource	Resource layout	Mediator	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM	DeM
		Engine Repository	DeM	DeM	DeM	DeM	DeM	DeM	StM	DeM	StM	StM	StM	DeM	DeM	StM	DeM	StM	StM	StM	DeM	DeM	DeM	DeM	DeM	DeM
		Workflow Repository	ExM	ExM	ExM	ExM	ExM	ExM	StM	StM	DeM	StM	ExM	ExM	DeM	StM	StM	DeM	StM	ExM	ExM	DeM	StM	DeM	ExM	DeM
		Engine execution	DeM	DeM	DeM	DeM	DeM	DeM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	CoM	DeM	DeM	DeM	DeM
Interface	Backend	CLI							X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		API	X	X	X	X	X	X																		
	Frontend	General	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Specific																								

Table 4.15: Existing and proposed WESA architectures, where signs have the same purpose as in the case of table 4.12.

these the instance layout of this solution can be defined as $h = \overline{\{(RE, E), (A, RW)\}}$. By default, the Gria Service is hosted on a single dedicated machine ($\delta_M = 1$, $\xi_M = DeM$), access is provided to multiple previously installed workflow engines hosted on dedicated machines ($\delta_{RE}, \delta_E \geqq 1; \xi_{RE}, \xi_E = DeM$). Finally, since workflows are provided by the applications, $\delta_{RW} = r$ and $\xi_{RW} = ExM$.

Engine path is (RE, E) in which case engine staging is irrelevant (see lemma A.3). Actually, since $(RE, E) \in h$ there is no physical engine transfer. Workflows by default are passed to the workflow engine via the mediator $\{(RW, M), (M, E)\}$, but if the workflow engine supports to receive workflows from third parties, it can be transferred directly from the application $\{(A, E)\}$. In the case of the former,

Architecture	Overhead	Overhead scalability	Latency	Latency scalability
Gria v1	$\frac{rl_wK}{\min\{r,\delta_E\}}$	$O(rl_w)$	$\frac{rl_wK}{\min\{r,\delta_E\}}$	$O(rl_w)$
Gria v2a	$\frac{rl_wK}{\min\{r,\delta_E\}} + \frac{rs_bK}{\min\{r,\delta_E\}} + + \frac{rl_bK}{\min\{r,\delta_E\}} - l_bK$	$O(rl_w + rl_b)$	$\frac{rl_wK}{\min\{r,\delta_E\}} + \frac{rs_bK}{\min\{r,\delta_E\}}$	$O(rl_w + r)$
Gria v2b	$\frac{rl_wK}{\min\{r,\delta_E\}} + 2\frac{rl_bK}{\min\{r,\delta_E\}} - -l_bK$	$O(rl_w + rl_b)$	$\frac{rl_wK}{\min\{r,\delta_E\}} + \frac{rl_bK}{\min\{r,\delta_E\}}$	$O(rl_w + rl_b)$
Gria v3	$\frac{rl_wK}{\min\{r,\delta_E\}} + 2rl_bK + + \frac{rl_bK}{\min\{r,\delta_E\}} - l_bK$	$O(rl_w + rl_b)$	$\frac{rl_wK}{\min\{r,\delta_E\}} + 2rl_bK$	$O(rl_w + rl_b)$
Gria v4	$2rl_wK$	$O(rl_w)$	$2rl_wK$	$O(rl_w)$
Gria v5a	$2rl_wK + \frac{rs_bK}{\min\{r,\delta_E\}} + + \frac{rl_bK}{\min\{r,\delta_E\}} - l_bK$	$O(rl_w + rl_b)$	$2rl_wK + \frac{rs_bK}{\min\{r,\delta_E\}}$	$O(rl_w + r)$
Gria v5b	$2rl_wK + 2\frac{rl_bK}{\min\{r,\delta_E\}} - -l_bK$	$O(rl_w + rl_b)$	$2rl_wK + \frac{rl_bK}{\min\{r,\delta_E\}}$	$O(rl_w + rl_b)$
Gria v6	$2rl_wK + 2rl_bK + + \frac{rl_bK}{\min\{r,\delta_E\}} - l_bK$	$O(rl_w + rl_b)$	$2rl_wK + 2rl_bK$	$O(rl_w + rl_b)$

Table 4.16: Performance characteristics of the Gria Service based architectures in the case where $(E, J) \notin h$. Cases marked with a and b are representing the same as in table 4.14.

Architecture	Overhead	Overhead scalability	Latency	Latency scalability
Gria v1, v2	$\frac{rl_wK}{\min\{r,\delta_E\}} + \frac{rl_bK}{\min\{r,\delta_E\}} - -l_bK$	$O(rl_w + rl_b)$	$\frac{rl_wK}{\min\{r,\delta_E\}}$	$O(rl_w)$
Gria v3	$\frac{rl_wK}{\min\{r,\delta_E\}} + 2rl_bK - -l_bK$	$O(rl_w + rl_b)$	$\frac{rl_wK}{\min\{r,\delta_E\}} + 2rl_bK$	$O(rl_w + rl_b)$
Gria v4, v5	$2rl_wK + \frac{rl_bK}{\min\{r,\delta_E\}} - -l_bK$	$O(rl_w + rl_b)$	$2rl_wK$	$O(rl_w)$
Gria v6	$2rl_wK + 2rl_bK - -l_bK$	$O(rl_w + rl_b)$	$2rl_wK + 2rl_bK$	$O(rl_w + rl_b)$

Table 4.17: Performance characteristics of the Gria Service based architectures in the case where $(E, J) \in h$.

workflow staging is non pipelined, while in the case of the later workflow staging is irrelevant. Three types of bulk data paths can be implemented via this solution: (A, J), (A, E, J), (A, M, E, J). The latter is the default approach, but the former two can be applied if it is supported by a particular workflow job or workflow engine

to receive data from third party machines. In the case of (A, J) bulk data staging is irrelevant, in the case of (A, E, J), depending on the particular engine, both bulk data staging approaches can be realised, while in the case of (A, M, E, J) bulk data staging is not pipelined. All possible architectures that can be realised based on the Gria service are included in table 4.15 along with the proposed architectures. Structure and data flow combinations of the Gria based solution is illustrated in figure 4.4. Performance characteristics of the Gria based architectures can be seen in table 4.16 and 4.17.

4.4.2 Proposed WESA solutions

The WESA architecture analysis described in section 4.3, proposes several different architectures in different cases. Based on section 4.3.1, frontend interface of the proposed architectures should be general meaning that the data type and number of input and output parameters of a workflow should not be restricted. Furthermore, in order to provide architectures that are easily extendible with further workflow engines, the backend interface is recommended to be CLI. Section 4.3.5 identifies several proposed structure and data flow type combinations which are defined in table 4.12. Table 4.13 summarises which structure and data flow type combinations are recommended in 18 different cases. Having these, all proposed architectures are specified in table 4.15 along with the architectures of the Gria service based solution. The recommended WESA solution should realise one or more of the proposed architectures depending on what cases it will be used in.

4.4.3 Comparison of existing and proposed WESA solutions

While the Gria service based architectures invoke a previously installed workflow engine in a service oriented manner, many of the proposed solutions can also submit the workflow engines to computational Grid resources distributing the load on bulk data transfer. In the case of workflows of which jobs are executed locally to their workflow engines, architectures of PC1-PC7 are proposed, which submit the workflow engines to Grid resources. While engine transfer time the of Gria based architectures is always 0, it is linear with the number of simultaneous requests and engine size in the case of the above proposed solutions. However, overhead on bulk data transfer of Gria based architectures is always linear with the number of simultaneous requests and bulk data amount, in the case of the above proposed architectures it is always 0. Since bulk data amount in the case of large-scale workflows is significantly greater than engine size, overhead of the proposed architectures is less then the overhead of the Gria based architectures and is independent of bulk data amount.

In the case of workflows of which jobs are executed remotely to their workflow engines and can transfer bulk data directly to/from the application (i.e using GridFTP or other data transfer protocol), the architectures of PC15-PC17 are proposed. These utilise previously installed workflow engines and transfer bulk data directly between the application and the workflow job. Therefore, overhead on both engine and bulk data transfer is always 0, just like in the case of the Gria based architecture Gria v1 and v4.

In the case of workflows of which jobs are executed remotely to their workflow engines and can transfer bulk data only via their workflow engines, architectures of PC8-PC14 are proposed, which also submit the workflow engines to the Grid.

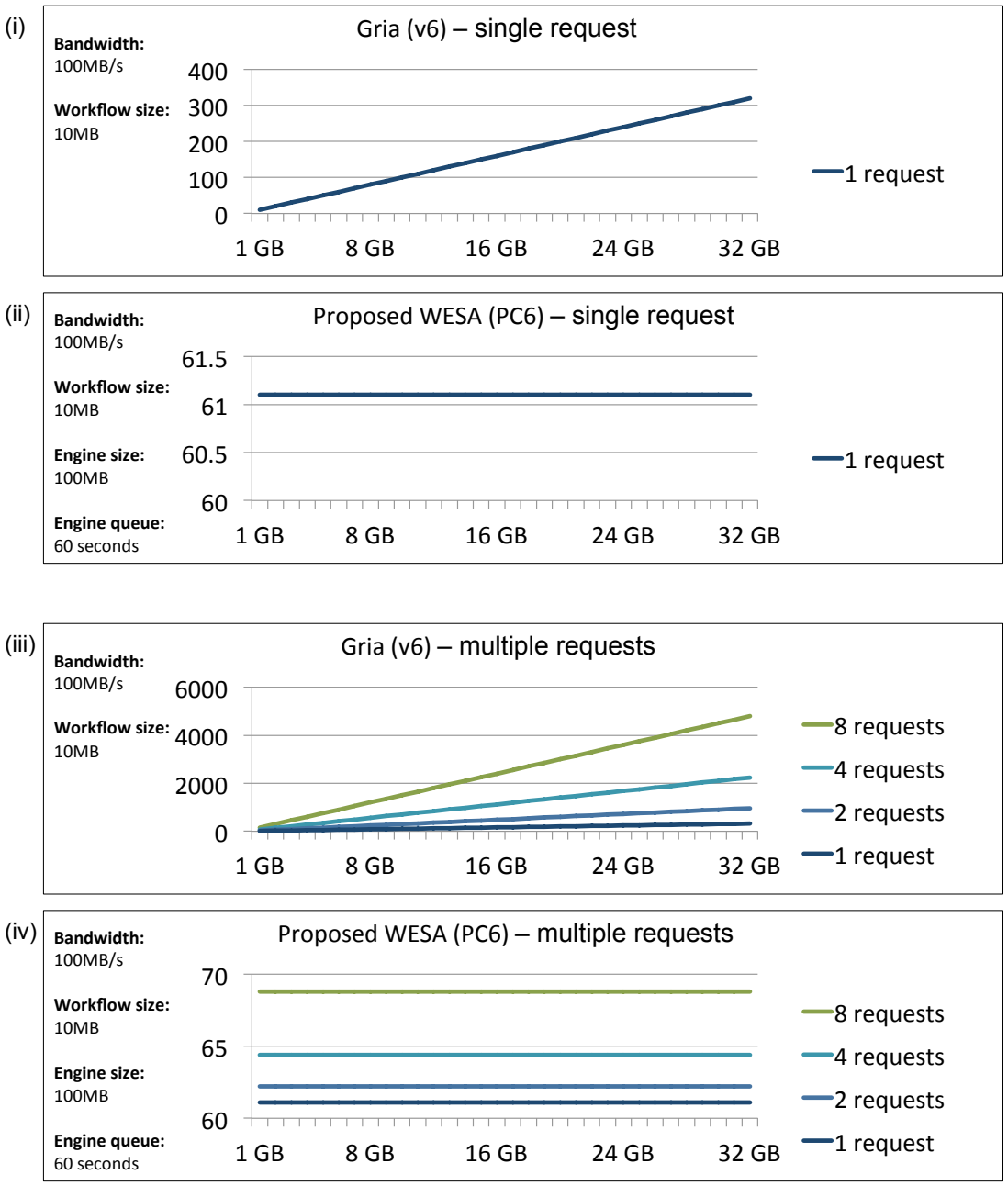

Figure 4.5: Overall overhead predictions of Gria (v6) and Proposed WESA (PC6) architectures in seconds in the function of bulk data size, where graph (i) and (ii) represent single, graph (iii) and (iv) represent multiple request executions.

Overhead is linear with the number of simultaneous requests and engine size in the case of these proposed solutions. Overhead on bulk data transfer of the Gria based architectures linearly increases with both the number of simultaneous requests and bulk data amount. However, in the case of the above proposed architectures if the workflow engine supports pipelined transfer it is always 0, while if the workflow engine does not support pipelined transfer then it is linear with bulk data amount but independent of the number of simultaneous requests. Therefore, overall overhead of the proposed architectures is less then the overall overhead of the Gria based architectures in the case of large numbers of multiple requests and is independent of bulk data amount if pipelined transfer is supported by the workflow engine. For detailed performance figures see table 4.14, 4.16 and 4.17.

Overhead predictions of the Gria (v6) architecture and the Proposed WESA (PC6) architecture are illustrated on figure 4.5. The graphs represent the overall overhead performance formulas provided for these architectures in the case of a single and multiple requests. The graphs were generated based on the overhead formulas defined in table 4.14 and 4.17 for the case where workflow jobs are executed locally to the workflow engines. The performance predictions are based on a network with 100MB/s bandwidth and 10MB Workflow size and 100MB Engine size. Overhead of the Gria (v6) architecture increases linearly with bulk data size, as shown on graph (i). In the case of 32GB, overhead is above 300s (5 minutes). This is resulted by the fact that first bulk data is transferred to the Gria service machine and then transferred further to the application machine. In comparison, as graph (ii) shows, overhead of the Proposed WESA architecture (PC6) is 61.1s. This is the amount of time required for the workflow engine to be transferred from the Engine repository machine to the machine where it is executed (1s) with additional engine queuing (60s) and also including the amount of transfer time required for the workflow description to be transferred from the Workflow repository machine to the workflow engine (0.1s). Since after this point bulk data is transferred directly between the

application machine and the job machine, overhead is constant and independent of bulk data size. Graph (iii) and (iv) show that overhead of both architectures increase linearly with request number. However, while in the case of the Gria (v6) architecture with 8 simultaneous requests and 32GB bulk data size overhead is nearly 5000s (about 83 minutes), overhead of the Proposed WESA architecture (PC6) with the same number of parallel requests is below 70s.

Another difference is that in the case of the Gria based architectures engines are accessed via APIs, which means that programming knowledge is required to add a new workflow engine to an existing WESA. In the case of the proposed concepts, engines are connected via CLI. This means that user level knowledge is sufficient to add a new engine to the system, provided that a special user interface, such as the GEMLCA administration portlet [99], is available for describing CLIs.

4.5 Implementation

Several WESA architectures were implemented based on the GEMLCA (see description in section 2.5) application repository and submitter. Command-line workflow engines, just like DRCs or legacy applications, can be published via GEMLCA, without code re-engineering and can be executed by GEMLCA on computational Grid resources. Frontend interface of GEMLCA is general since it does not restrict the number or type of parameters that can be specified on engine execution. Backend interface is CLI, since workflow engines are accessed via their command line interfaces.

Four workflow engines Kepler, MOTEUR, Taverna, and Triana were deployed in the GEMLCA application repository. The engines were placed to a GridFTP storage machine and wrapper scripts were created which are able to download, parametrise,

and execute them. In order to make the workflow engines accessible, these wrapper scripts were deployed to the GEMLCA service via the GEMLCA Administration Portlet. Figure 4.6 shows how the Taverna engine can be exposed using this Portlet.

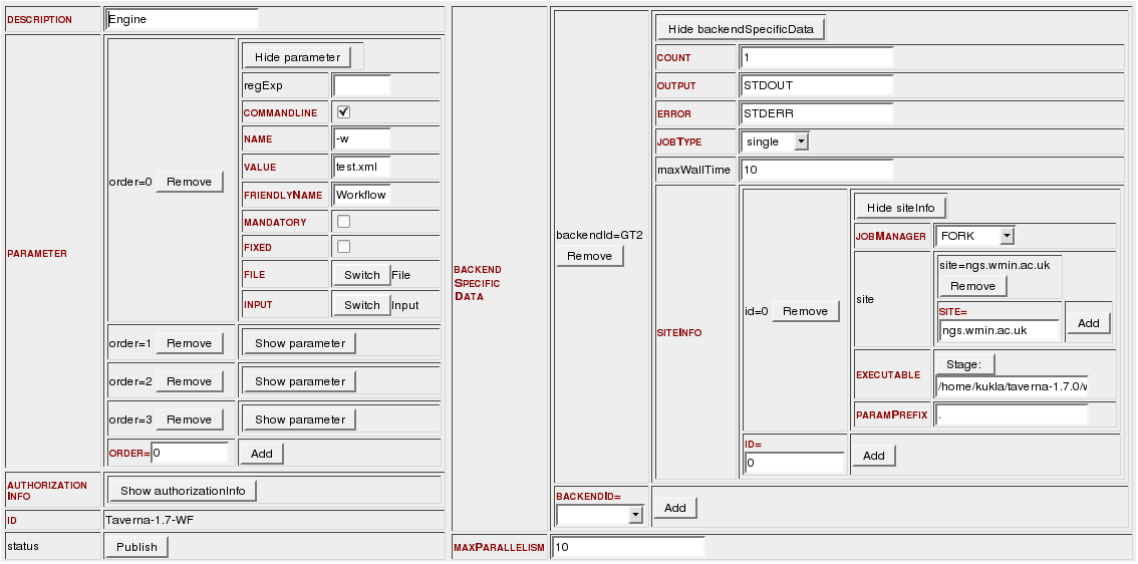

Figure 4.6: Deploying Taverna workflow engine using the GEMLCA Administration Portlet

In order to enable GEMLCA to expose and execute not only workflow engines but workflow as well, GEMLCA was extended with a so called Generic Interpreter Backend (GIB). GIB allows to deploy and connect two different legacy applications: an interpreter and an interpreted application. The interpreter application (workflow engine) receives the interpreted application (workflow) as an input file and executes it transparently. This concept is used to deploy and connect workflow engines and workflows in GEMLCA. Using the GEMLCA Administration Portlet, a so called engine administrator can deploy different workflow engines as interpreter applications, while workflow developers can deploy workflows as interpreted applications and select which workflow engine can execute it.

Figure 4.7 illustrates how the GEMLCA based WESA solution implements PC10.

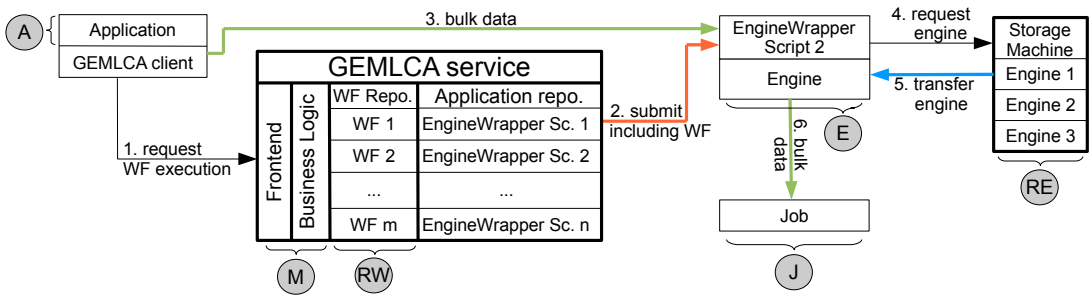

Figure 4.7: Implementation of WESA PC10 based on GEMLCA, where black arrows represent control data, blue arrow represents engine transfer, yellow arrow represents workflow transfer, and green arrows represent bulk data transfer.

The GEMLCA service realises the mediator and also the workflow repository, and a GridFTP storage machine realises the engine repository. This means that the mediator is coupled with the workflow repository and the engine repository is decoupled. The application passes a request to the local GEMLCA client. The request includes which GEMLCA service to invoke, which workflow to execute, the computational resource where the workflow engine should be executed, and the workflow arguments. The GEMLCA client submits a request to the GEMLCA service to execute the selected workflow. The GEMLCA service knows which workflow engine this workflow should be executed by. It submits the wrapper script of this engine to the desired location along with the selected workflow descriptor. Next, bulk data is transferred from the GEMLCA client to this machine. The wrapper script retrieves the appropriate workflow engine from the engine repository and starts the execution of the workflow engine. Finally, the workflow engine transfers bulk data to the job that needs to process it. Selecting a workflow for execution from the GEMLCA application repository is not necessary, the application can also provide a workflow for execution. In this case the workflow is transferred directly by the GEMLCA client from the application machine to engine. Therefore the GEMLCA based solution also implements PC5.

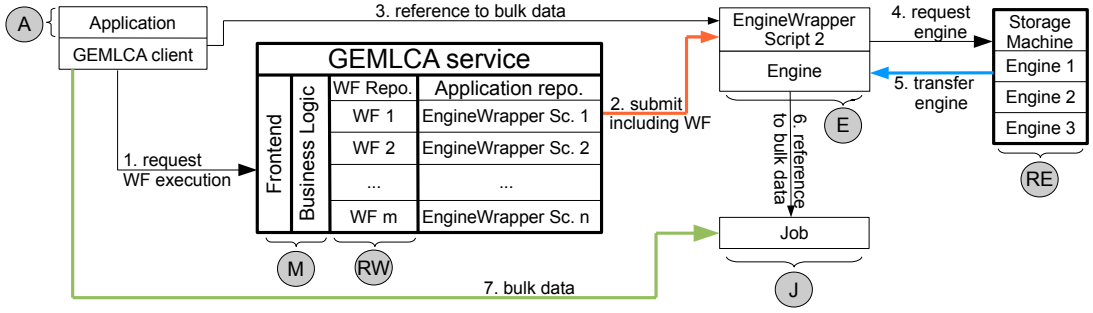

Figure 4.8: Implementation of WESA PC3 based on GEMLCA

In the case when it is possible for the job to gather data directly from the machine of the application (i.e. the application machine hosts a GridFTP server), only a reference to the bulk data should be passed via GEMLCA and bulk data should be transferred directly between the application and the job. See illustration on figure 4.8. This way PC3 (workflow is in GEMLCA application repository) and PC12 (workflow is provided by the application) also can be implemented.

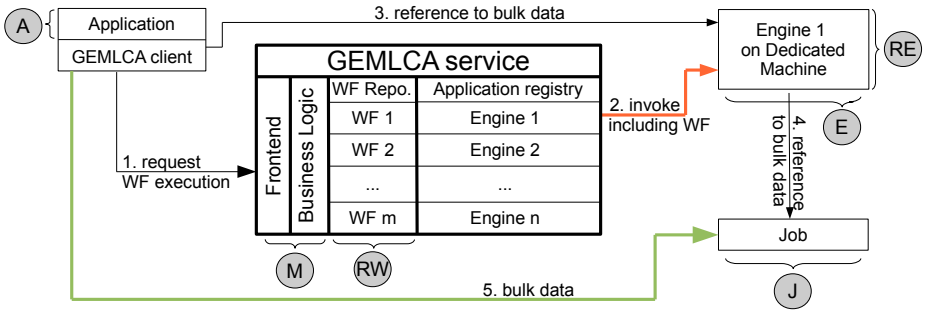

Figure 4.9: Implementation of WESA PC16 based on GEMLCA

GEMLCA supports not only the submission of legacy applications, but it also supports the remote execution of previously installed applications hosted on dedicated machines. The four workflow engines were also deployed on dedicated machines at the local NGS cluster of the University of Westminster. If the job can gather bulk data from the application machine then PC16 (workflow is in GEMLCA application repository) and PC17 (workflow is provided by the application) can also

be implemented via GEMLCA. See illustration in figure 4.9.

The solution was tested on the UK NGS (based on Globus) and on Gilda (based on gLite) with three types of workflows:

- (a) workflows of which jobs are executed locally to the workflow engine,
- (b) workflows of which jobs are remote web services, and
- (c) workflows of which jobs are submitted to Globus based computational resources.

Case (a) was tested with Kepler, MOTEUR, Taverna, and Triana engines on both Globus and gLite middleware. Case (b) was tested with Taverna on both Globus and gLite middleware. Case (c) was tested with Taverna on Globus middleware. Taverna submitted jobs to other computational resources using the command line submitter tools provided on the machine that executed the Taverna engine. Note that these tools are available on most NGS sites, but typically not available on EGEE sites.

Software dependencies of the workflow engines have to be linked statically, if they are not provided on the computational resource where they are executed. Firewall settings of the different computational resources may limit the functionality of the workflow engines and disable them to submit jobs to other computational resources.

4.6 Summary

This chapter proposed 18 WESA architectures to enable applications to execute workflows of different types independently of what workflow system they were designed in originally. Note that this application can be a Grid application, a simple

client which can be used by scientists to execute different types of workflows even if they are not familiar with the workflow system they were designed in, or any kind of application that needs a specific functionality that is provided by a given workflow.

The described analysis not only compares proposed architectures and architectures of existing solutions, but also compares numerous other possibilities at the level of data flow. This data flow analysis also can be utilised in special scenarios which are not addressed by this research. Note that the analysis is only based on data flow, computational load generated by the different workflow engines is out of the scope of this thesis. This can be addressed by future work.

The only existing WESA solution is based on the Gria service which makes a small set of workflow engines available via Web/Grid services. In this concept workflow engines are deployed on dedicated machines and invoked by Gria. By default this approach transfers bulk data from the application via the Gria service to the workflow engine that transfers it further to the workflow job that processes it. Transferring large amounts of bulk data this way is not recommended, since both the Gria service and the machine that hosts the workflow engine can bottleneck the transfer in the case of large number of requests. Alternatively, a reference to the bulk data should be transferred and bulk data should be gathered by the job that processes it directly. However, this is can be only realised if the job and the application can directly exchange data which cannot be guaranteed in general. If this is possible and jobs are executed remotely from the workflow engine, then the proposed architectures recommend a similar concept to the Gria service based solution. In all other cases, the proposed architectures distribute workflow engines between the available computational machines and bulk data is exchanged between the engines and the application directly. Although, the distribution of workflow engines increases overhead and latency, after the engine is in place, bulk data can flow directly. In contrast to the Gria based solution, this approach is scalable even

in the case of large number of simultaneous requests.

The GEMLCA based reference implementation realises 6 of the proposed architecture sets. It supports the execution of Kepler, MOTEUR, Taverna, and Triana workflows. Dynamic distribution of workflow engines on computational Grid resources can be realised only on resources where the software dependencies of a given engine are fulfilled. If this is not provided all required libraries have to be statically linked. In the case of workflows where jobs have to be submitted to remote resources, submission has to be supported on the given computational resource where the engine is executed. Although existing Grid middleware products can support this approach, due to administrative limitations (i.e. firewall restrictions) in the case of some Grid infrastructures (i.e. EGEE) this is not supported. Similarly to DASGs and DASWs, the GEMLCA based WESA solution is also easily extendible with any workflow engine that has a command line interface, which is provided in most cases. This can be achieved using the GEMCLA administration portlet that enables the description of the workflow engine CLI via a simple graphical interface without code re-engineering. Furthermore, it is a general solution, because GEMCLA does not restrict the number and type of input parameters that can be passed to the engine. Although, the parameters have to be represented either as command line arguments or files, this does not mean that all data to the workflow has to be passed as command line arguments or files. If a particular workflow job needs to gather data form a given Web Service, the HTTP end point of the Web Service can be passed as an argument and the workflow job can connect the Web Service during workflow execution.

Chapter 5

Heterogeneous Workflow Execution Solutions for Workflows (WESW) - workflow nesting

Concepts of WESWs and WESAs are very similar. The main difference is that WESWs provide service not for applications but for workflows. See illustration in figure 5.1. WESWs enable interoperation of heterogeneous workflows at the level of workflow nesting.

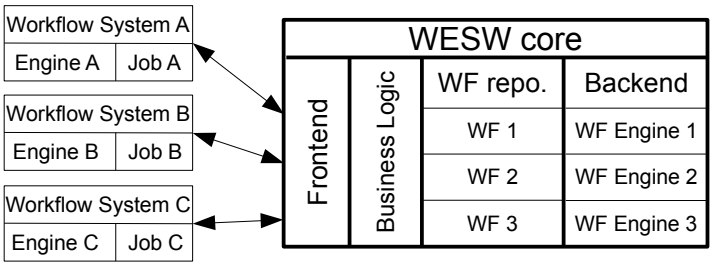

Figure 5.1: WESW concept

5.1 Key WESW properties and requirements

Six key properties of WESWs: (i) generality, (ii) extendibility, (iii) overhead, (iv) latency, (v) scalability, and (vi) invocation were taken under consideration. Properties (i-v) are important for the same reasons as in the case of WESAs described in section 4.1.

Invocation The property *invocation* is similar to the previously defined data access property for DASWs. (See section 3.1.) However, in the case of WESWs, rather than accessing heterogeneous data, heterogeneous workflow engines are invoked. Workflow engine invocation (invocation for short), like data access, can be *static*, *semi-dynamic*, or *dynamic*. See illustration in figure 5.2. Static invocation means that the child workflow specified/selected and invoked before or after the parent workflow is executed. Semi-dynamic invocation means that the workflow is specified/selected before, but it is invoked during parent workflow execution. Dynamic invocation means that the child workflow is specified/selected and executed as part of the parent workflow. Static workflow invocation allows sequential execution of the parent and child workflows, but does not enable workflow nesting. Therefore it is not suitable for WESWs. Semi-dynamic invocation can be suitable in many cases, but dynamic invocation provides the greatest flexibility.

5.2 WESW architecture definition

To study possible approaches and identify optimal solutions, WESWs are investigated from five aspects: *structure*, *resources*, *data flow*, *interface*, and *integration*.

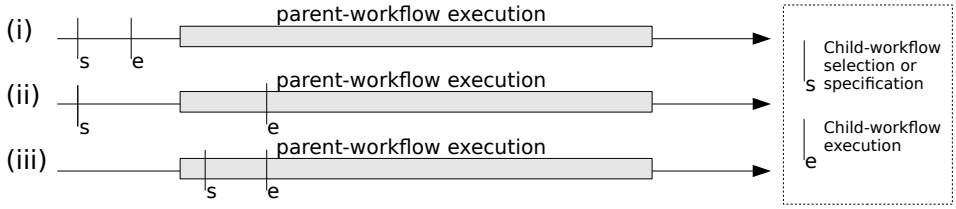

Figure 5.2: WESW Workflow invocation types: (i) static, (ii) semi-dynamic, and (iii) dynamic

These are defined in the followings.

5.2.1 WESW structure

In analogy with previously defined structures, WESW structure definition is also based on the general definitions introduced in chapter 2.

Definition 5.1 (WESW nodes and node types)

- A *parent engine node* represents a running workflow engine. This engine executes a workflow (so called parent workflow) that is to execute another workflow (a child workflow). Parent engine nodes belong to node type EP.
- A *parent job node* represents a task initiated by a parent workflow engine node. It either generates or processes data that is to be exchanged with a child job node (see below). Parent job nodes belong to node type JP.
- A *child engine node* receives a workflow engine from an engine repository and executes it locally. After this point it represents the running workflow engine which enacts a child workflow. These nodes belong to type EC.
- A *child job node* is a task that is initiated by a child engine node. It either generates or processes data that is to be exchanged with a parent job node. Child job nodes belong to node type JC.

Mediator, *engine repository*, and *workflow repository* nodes and node types represent the same concepts as definition 4.1 describes them in the case of WESAs. Let $\mathcal{T}' = \{M, RE, RW, EC\}$ be the set of *core WESW node types*, $\mathcal{T}'' = \{EP, JP, JC\}$ be the set of *external WESW node types*, and $\mathcal{T} = \{EP, JP, M, RE, RW, EC, JC\}$ be the set of *all WESW node types*. See illustration on figure 5.3.

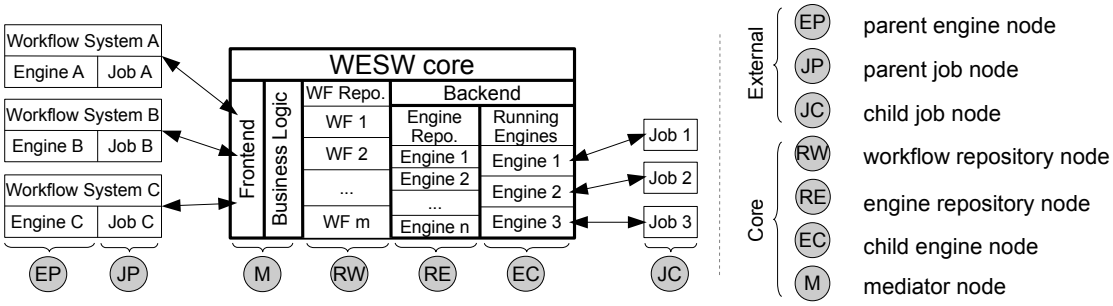

Figure 5.3: WESW node types

Definition 5.2 (WESW Instance)
Let a *WESA instance* be a set of $|\mathcal{T}| = 7$ nodes, where each node belongs to a different node type of $\{EP, JP, M, RE, RW, EC, JC\}$.

Definition 5.3 (Bijection between WESW node types and instances)
Let $\forall i \in [1..r]$: let $\mathcal{N}_i := \{EP_i, JP_i, M_i, RE_i, RW_i, EC_i, JC_i\}$ be the ith WESW instance, where $\varphi_i(EP) = EP_i$, $\varphi_i(JP) = JP_i$ $\varphi_i(M) = M_i$, $\varphi_i(RE) = RE_i$, $\varphi_i(RW) = RW_i$, $\varphi_i(EC) = EC_i$, and $\varphi_i(JC) = JC_i$.

In the case of WESWs there are 7 node type sets $\mathcal{N}_{EP}$, $\mathcal{N}_{JP}$, $\mathcal{N}_M$, $\mathcal{N}_{RE}$, $\mathcal{N}_{RW}$, $\mathcal{N}_{EC}$, $\mathcal{N}_{JC}$ and r nodes in each type set. A WESW node matrix of instances and types can be constructed as illustrated in table 3.1. Furthermore, both $\bigcup_{i=1}^{r} \mathcal{N}_i$ and $\bigcup_{t \in \mathcal{T}} \mathcal{N}_t$ are equal to the set of all WESW nodes, $\mathcal{N}$ and $|\mathcal{N}| = 7r$.

Instances	Node types: $\mathcal{N}_{EP}$	$\mathcal{N}_{JP}$	$\mathcal{N}_{M}$	$\mathcal{N}_{RE}$	$\mathcal{N}_{RW}$	$\mathcal{N}_{EC}$	$\mathcal{N}_{JC}$
$\mathcal{N}_1$	EP_1	JP_1	M_1	RE_1	RW_1	EC_1	JC_1
$\mathcal{N}_2$	EP_2	JP_2	M_2	RE_2	RW_2	EC_2	JC_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\mathcal{N}_r$	EP_r	JP_r	M_r	RE_r	RW_r	EC_r	JC_r

Table 5.1: WESW node matrix

5.2.2 WESW structure

WESW structure layout is also based on instance and type layout.

Definition 5.4 (WESW instance layout)
The set of all possible WESA instance layouts is represented by $\mathcal{L}_I$ and equals to the set of all possible type layouts on domain $\mathcal{T}$ (see definition 2.11).

Definition 5.5 (WESW type layout)
The set of all possible WESW type layouts is represented by $\mathcal{L}_T$ and equals to the set of all possible type layouts on domain $\mathcal{T}$ (see definition 2.13).

Definition 5.6 (WESW structure layout)
The set of all possible WESW structure layouts is represented by $\mathcal{L}_S$ and equals to the set of all possible structure layouts on domain $\mathcal{T}$ (see definition 2.15).

5.2.3 WESW data flow

Definition 5.7 (WESW data types)
Data between distributed nodes of $\mathcal{N}$ can flow in various ways. To identify the different possibilities, the same four kinds of data are distinguished like in the case

of WESAs:

- *bulk data* is the data-set that needs to be transferred between (JP_i) and (JC_i);
- *engine data* is the workflow engine executable itself that needs to be transferred from (RE_i) to (EC_i);
- *workflow data* is the workflow descriptor and all further data (fix parameters, job executables, etc) that need to be transferred from (RW_i) to (EC_i); and
- *control data* is the set of information that includes all further data transferred between the nodes. This consists of a small number of requests which are necessary to exchange in order to enable a workflow to execute another.

For the same reasons as in the case of WESAs, only three kinds of data flow are considered here: engine data flow, workflow data flow and bulk data flow.

Definition 5.8 (WESW engine data flow)
In the case of WESWs four engine path types are distinguished: when the engine is transferred directly (RE, EC), via the mediator (RE, M, EC), via the parent workflow engine (RE, EP, EC), and via both the mediator and the parent workflow engine (RE, M, EP, EC). Let $\mathcal{E}_P := \{(RE, EC), (RE, M, EC), (RE, EP, EC), (RE, M, EP, EC)\}$ be the *set of engine path types* and let $\mathcal{E}_S = \{Pip, \neg Pip\}$ be the *set of engine staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined engine staging.

Definition 5.9 (WESW workflow data flow)
In the case of WESWs four workflow path types are distinguished: when the engine is transferred directly (RW, EC), via the mediator (RW, M, EC), via the parent workflow engine (RW, EP, EC), and via both the mediator and the parent workflow

engine (RW, M, EP, EC). Let $\mathcal{W}_P := \{(RW, EC), (RW, M, EC), (RW, EP, EC), (RW, M, EP, EC)\}$ be the *set of workflow path types*. Let $\mathcal{W}_S = \{Pip, \neg Pip\}$ be the *set of engine staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined workflow staging.

Definition 5.10 (WESW bulk data flow)

Seven bulk data flow path types are distinguished in the case of WESWs: bulk data can be transferred directly (JP, JC), via the child engine (JP, EC, JC), via the parent engine (JP, EP, JC), via the mediator (JP, M, JC), via both parent and child engine (JP, EP, EC, JC), via both the mediator and child engine (JP, M, EC, JC), via both the parent engine and the mediator (JP, EP, M, JC), and via all the parent engine, the mediator, and the child engine (JP, EP, M, EC, JC). Let $\mathcal{B}_P := \{(JP, JC), (JP, EC, JC), (JP, EP, JC), (JP, M, JC), (JP, EP, EC, JC), (JP, M, EC, JC), (JP, EP, M, JC), (JP, EP, M, EC, JC)\}$ be the *set of bulk data path types* and let $\mathcal{B}_S = \{Pip, \neg Pip\}$ be the *set of bulk data staging types*, where Pip represents pipelined, while $\neg Pip$ represents non pipelined bulk data staging.

Definition 5.11 (WESW data flow types)

Having these, let $\mathcal{DF} := \mathcal{E}_P \times \mathcal{E}_S \times \mathcal{W}_P \times \mathcal{W}_S \times \mathcal{B}_P \times \mathcal{B}_S$ be the *set of WESA data flow types*.

5.2.4 WESW resources

Definition 5.12 (WESW resource layout)

Definition 5.1 identified three external node types (EP, JP, JC) and the same core node types as definition 4.1 for WESAs, except for child engine execution (EC) which in the case of WESAs is called engine execution (E). For the same reasons

described in definition 4.12 the same resource types can be mapped to the core note types as in the case of WESAs, and external machine resources are mapped to all external node types. Therefore, the set of all possible WESW resource layouts is defined as:

$$\begin{aligned}\mathcal{RL} := \{\xi \in \mathcal{L}_R(\mathcal{T}) \| \xi_M \in \{DeM, ExM\} \wedge \xi_{RE} \neq CoM \wedge \xi_{RW} \neq CoM \wedge \\ \wedge \xi_{JC} \neq StM \wedge \xi_{EP}, \xi_{JP}, \xi_{JC} = ExM\}\end{aligned} \tag{5.1}$$

5.2.5 WESW interface

Definition 5.13 (WESW interfaces)
Let $\mathcal{I}_F := \{Gen, Spe\}$ be the *set of frontend interface types*, let $\mathcal{I}_B := \{CLI, API\}$ be the *set of backend interface types*, and let $\mathcal{IN} := \mathcal{I}_F \times \mathcal{I}_B$ be the *set of interface types*.

5.2.6 WESW integration

Definition 5.14 (WESW Subject of integration)
The *subject of integration* is the particular part of the parent workflow system that will be able to communicate with the child workflow engines. Let $\mathcal{G}_S := \{AuT, WEn\}$ be the *set of integration subjects*. Depending on which part of the system will be enhanced with this capability, the integration can be realized at: (i) *workflow engine* or (ii) *auxiliary tool* level. Both integration types are partially identical to the ones described in section 3.2. The only difference is that the subject of integration is extended with a tool, that is able to communicate with heterogeneous workflow engines, rather that with heterogeneous data resources and workflow editor integration type is excluded here.

5.2.7 WESW architecture and solution

Definition 5.15 (Set of possible WESW architectures)
The *set of possible WESW architectures* consists of elements of the Cartesian product of sets of possible structures, data flows, interfaces, and integrations as:

$$\begin{aligned}\mathcal{AR} := \{((h,\delta),\xi,(q_e,s_e,q_w,s_w,q_b,s_b),(i_f,i_b),g_s)) \in \mathcal{L}_S \times \mathcal{L}_R \times \mathcal{DF} \times \mathcal{IN} \times \mathcal{G}_S \times \mathcal{L}_R \| \\ q_e, q_w, \text{ and } q_b \text{ are acyclic path layouts based on instance layout } h \wedge \quad (i) \\ \wedge \forall (t_1,t_2) \in h \in \mathcal{T} : \xi_{t_1} = \xi_{t_2}\}. \quad (ii)\end{aligned} \tag{5.2}$$

Note that conditions are needed for the same reasons as described in the case of DASG architectures in definition 2.40.

Definition 5.16 (WESW solution)
A WESW solution is a set of WESW architectures. With other words, it is a not empty subset of $\mathcal{AR}$.

5.3 WESW architecture analysis

5.3.1 WESW generality, extendibility, and invocation

In order to provide general and easily extendible architectures, general frontend interface, command line backend interface, and centralised mediator hosted on a dedicated machine are recommended, for the same reasons described in the case of WESAs in section 4.3.1.

In terms of subject of integration, auxiliary tool (AuT) level integration provides only static workflow engine invocation, that is not suitable in many cases. Workflow

engine (WEn) level integration, however, provides both semi-dynamic and dynamic workflow engine invocation, that is needed in most use cases. Therefore, workflow engine level integration is recommended.

5.3.2 WESW performance

The aim of the performance analysis is to compare overhead, latency, and scalability of different WESW architectures and show how these values vary with bulk data volume, workflow size, engine size, and number of simultaneous requests. The performance comparison is based on WESW scenarios where $r \in \mathbb{N}^+$ different parent workflow jobs hosted by different machines initiate the execution of r different child workflows. The first job of each child workflow is a job that receives data from the parent workflow job that initiated the execution. Parent engines are always hosted on r different machines and the same is true for the parent workflow jobs. Scenarios are represented as the elements of the set defined below.

Definition 5.17 (WESW scenarios)

$$
\begin{aligned}
&\text{Let } \mathcal{AS} := \{((h,\delta),\xi,q_e,q_w,q_b,w_e,s_e,s_w,s_b,l_e,l_w,l_b,r) \in \\
&\qquad \in \mathcal{L}_S \times \mathcal{L}_R \times \mathcal{E}_P \times \mathcal{W}_P \times \mathcal{B}_P \times \mathbb{R}_0^+ \times (\mathbb{N}^+)^7 \| \\
&\quad \delta_{EP}, \delta_{JP} = r \wedge && (i) \\
&\wedge q_e, q_w \text{ and } q_b \text{ are acyclic path layouts based on } h \wedge && (ii) \\
&\wedge \forall (t_1,t_2) \in h : \xi_{t_1} = \xi_{t_2} \wedge && (iii) \\
&\wedge \xi_{EC} \neq CoM \Rightarrow w_e = 0 \wedge && (iv) \\
&\wedge \forall t \in \mathcal{T} \backslash \{EP, JP\} : r \equiv 0 \mod \delta_t \wedge && (v) \\
&\wedge l_e \equiv 0 \mod s_e \wedge l_w \equiv 0 \mod s_w \wedge l_b \equiv 0 \mod s_b\}. && (vi)
\end{aligned}
\tag{5.3}
$$

be the *set of analytical WESW scenarios*. Let $a = ((h, \delta), \xi, q_e, q_w, q_b, w_e, s_e, s_w, s_b, l_e, l_w, l_b, r) \in \mathcal{AS}$ be an analytical scenario. Parameters of a and conditions represent the same concepts as in the case of analytical WESA scenarios defined in definition 4.18.

Definition 5.18 (WESW scenario execution)
Since control flow is excluded from the model, similarly to WESA scenario execution, the analysis is based on engine, workflow and bulk data flow. $\forall i \in [1..r]$: let $e_i \in \mathcal{B}$ byte array represent the engine that executes workflow w_i (see below) and to be transferred from RE_i to EC_i, $w_i \in \mathcal{B}$ byte array represent the workflow that is to be invoked by the parent workflow EP_i and to be transferred from RW_i to EC_i, and $b_i \in \mathcal{B}$ byte array represent the bulk data that is to be transferred from JP_i to JC_i. A WESW scenario is executed in four steps:

1. engine transfer: $\forall i \in [1..r]$: e_i is transferred from RE_i to EC_i via path $\psi_i(q_e)$ simultaneously,
2. workflow transfer: $\forall i \in [1..r]$: w_i is transferred from RW_i to EC_i via path $\psi_i(q_w)$ simultaneously,
3. engine queuing: all engines are waiting w_e amount of time to be scheduled for execution,
4. bulk data transfer: $\forall i \in [1..r]$: the execution of the ith child workflow engine starts and b_i is transferred from JP_i to JC_i via path $\psi_i(q_b)$ simultaneously.

Engine flow can be represented as a simultaneous transfer (see definition 2.48), since the conditions of definition 5.17 ensure that $((h, \delta), q_e, s_e, l_e, r) \in \mathcal{D}_{st}$. Similarly, workflow and bulk data flow also can be represented as: $((h, \delta), q_w, s_w, l_w, r)$,

$((h, \delta), q_b, s_b, l_b, r) \in \mathcal{D}_{st}$ respectively. Having these the performance functions of a WESW scenario can be defined as follows.

Definition 5.19 (Performance of WESW engine and workflow transfer) Let $a := ((h, \delta), \xi, q_e, q_w, q_b, w_e, s_e, s_w, s_b, l_e, l_w, l_b, r) \in \mathcal{AS}$, and $\Gamma_e, \Gamma_w : \mathcal{AS} \to \mathbb{R}_0^+$ be functions for determining respectively engine workflow transfer time as:

$$\Gamma_e(a) := \tau_q((h, \delta), q_e, s_e, l_e, r) \qquad \Gamma_w(a) := \tau_q((h, \delta), q_w, s_w, l_w, r). \tag{5.4}$$

Note that definition 2.49 is applied to identify $\Gamma_e(a)$ and $\Gamma_w(a)$.

Definition 5.20 (Performance of WESW bulk data transfer) Let $a := ((h, \delta), \xi, q_e, q_w, q_b, w_e, s_e, s_w, s_b, l_e, l_w, l_b, r) \in \mathcal{AS}$, and $\Gamma_b, \Delta_b, \Theta_b : \mathcal{AS} \to \mathbb{R}_0^+$ be functions for determining respectively bulk data transfer time, overhead, and latency as:

$$\Gamma_b(a) := \tau_q((h, \delta), q_b, s_b, l_b, r) \tag{5.5}$$

$$\Delta_b(a) := \Gamma_b(a) - \chi((JP, JC) \notin h) k_b s_b K, \text{ and} \tag{5.6}$$

$$\Theta_b(a) := \epsilon_q((h, \delta), q_b, s_b, l_b, r). \tag{5.7}$$

Note that definition 2.49 is applied to determine $\Gamma_b(a)$ and $\Theta_b(a)$. $\forall i \in [1..r]$: transferring b_i directly between JP_i and JC_i takes $\tau_e(\lambda(b_i), (JP_i, JC_i))$ time. This value is 0 if $(JP, JC) \in h$ and $k_b s_b K$ otherwise. Overhead on bulk data transfer of a particular scenario is considered as this time subtracted from bulk data transfer time.

Definition 5.21 (Overall WESW performance functions) Let $\Gamma, \Delta, \Theta : \mathcal{AS} \to \mathbb{R}_0^+$ be functions for determining respectively execution time,

overhead, and latency of a scenario, where:

$$\Gamma(a) := w_e + \Gamma_e(a) + \Gamma_w(a) + \Gamma_b(a), \tag{5.8}$$

$$\Delta(a) := w_e + \Gamma_e(a) + \Gamma_w(a) + \Delta_b(a), \text{ and} \tag{5.9}$$

$$\Theta(a) := w_e + \Gamma_e(a) + \Gamma_w(a) + \Theta_b(a). \tag{5.10}$$

Because engine and workflow transfer always has to be performed before workflow execution, engine and workflow transfer times are always added to the overall latency and overhead of a scenario.

Definition 5.22 (Scalability of WESW data transfer)
Performance functions of any WESW scenario are characterised based on growth rates in function of l_w, l_e, l_b, and r. It is represented by the Bachmann–Landau (Big O) notation in analogy with all previous contributions.

5.3.3 WESW bulk data flow

Based definition 2.54 and the seven bulk data path layout types (see definition 5.10, 54 bulk data flow cases can be identified. These are listed in table 5.5-5.7. Restrictions implied by instance layout are also illustrated in the tables and representative cases (see definition 2.54) are marked with asterisks.

Based on definition 2.55, these cases can be divided into 9 groups. Performance properties are the same in each case of the same group. Transfer time, overhead, and latency values can be found in table 5.2, 5.3, and 5.4 respectively, where the formulas are based on the definition of $\Gamma_b(a)$, $\Delta_b(a)$, and $\Theta_b(a)$. Based on $\Delta_b(a)$ and $\Theta_b(a)$, the architectural conditions which determine scalability in terms of overhead and latency are identified for each group, these can be found in table 5.3, and 5.4. In particular, cases of group BG1 always provide 0 overhead and latency on bulk

Group	Transfer time ($\Gamma_b(a)$)
BG1	0
BG2	$\frac{rk_bs_bK}{\min\{r,\delta_{JC}\}}$
BG3	$\frac{rs_bK}{\min\{r\}} + \frac{rk_bs_bK}{\min\{r,\delta_{JC}\}}$
BG4	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rk_bs_bK}{\min\{r,\delta_M,\delta_{JC}\}}$
BG5	$\frac{rs_bK}{\min\{r,\delta_{EC}\}} + \frac{rk_bs_bK}{\min\{r,\delta_{EC},\delta_{JC}\}}$
BG6	$\frac{rs_bK}{\min\{r\}} + \frac{rs_bK}{\min\{r,\delta_{EC}\}} + \frac{rs_bK}{\min\{r,\delta_{EC},\delta_{JC}\}} + \frac{r(k_b-1)s_bK}{\min\{r,\delta_{EC},\delta_{JC}\}}$
BG7	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rs_bK}{\min\{r,\delta_M,\delta_{EC}\}} + \frac{rs_bK}{\min\{r,\delta_{EC},\delta_{JC}\}} + \frac{r(k_b-1)s_bK}{\min\{r,\delta_M,\delta_{EC},\delta_{JC}\}}$
BG8	$\frac{rs_bK}{\min\{r\}} + \frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rs_bK}{\min\{r,\delta_M,\delta_{JC}\}} + \frac{r(k_b-1)s_bK}{\min\{r,\delta_M,\delta_{JC}\}}$
BG9	$\frac{rs_bK}{\min\{r\}} + \frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rs_bK}{\min\{r,\delta_M,\delta_{EC}\}} + \frac{rs_bK}{\min\{r,\delta_{EC},\delta_{JC}\}} + \frac{r(k_b-1)s_bK}{\min\{r,\delta_M,\delta_{EC},\delta_{JC}\}}$

Table 5.2: Time of WESW bulk data transfer

Group	Overhead ($\Delta_b(a)$)	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
BG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG2	$\frac{rk_bs_bK}{\min\{r,\delta_{JC}\}} - k_bs_bK$	$\delta_{JC} \geqq r$	$\nexists$	$\nexists$	$\nexists$	$\delta_{JC} < r$
BG3	$rs_bK + \frac{rk_bs_bK}{\min\{r,\delta_{JC}\}} - -k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_{JC} \geqq r$	$k_b = 1$ $\delta_{JC} \geqq r$	$\nexists$	$\delta_{JC} < r$
BG4	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rs_bK}{\min\{r,\delta_M,\delta_{JC}\}} + + \frac{r(k_b-1)s_bK}{\min\{r,\delta_M,\delta_{JC}\}} - k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_M \geqq r$ $\delta_{JC} \geqq r$	$k_b = 1$ $\delta_M \geqq r$ $\delta_{JC} \geqq r$	$\nexists$	$\delta_M < r \vee$ $\vee \delta_{JC} < r$
BG5	$\frac{rs_bK}{\min\{r,\delta_{EC}\}} + \frac{rs_bK}{\min\{r,\delta_{EC},\delta_{JC}\}} + + \frac{r(k_b-1)s_bK}{\min\{r,\delta_{EC},\delta_{JC}\}} - k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_{EC} \geqq r$ $\delta_{JC} \geqq r$	$k_b = 1$ $\delta_{EC} \geqq r$ $\delta_{JC} \geqq r$	$\nexists$	$\delta_{EC} < r \vee$ $\vee \delta_{JC} < r$
BG6	$rs_bK + \frac{rs_bK}{\min\{r,\delta_{EC}\}} + + \frac{rs_bK}{\min\{r,\delta_{EC},\delta_{JC}\}} + + \frac{r(k_b-1)s_bK}{\min\{r,\delta_{EC},\delta_{JC}\}} - -k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_{EC} \geqq r$ $\delta_{JC} \geqq r$	$k_b = 1$ $\delta_{EC} \geqq r$ $\delta_{JC} \geqq r$	$\nexists$	$\delta_{EC} < r \vee$ $\vee \delta_{JC} < r$
BG7	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rs_bK}{\min\{r,\delta_M,\delta_{EC}\}} + + \frac{rs_bK}{\min\{r,\delta_{EC},\delta_{JC}\}} + + \frac{r(k_b-1)s_bK}{\min\{r,\delta_M,\delta_{EC},\delta_{JC}\}} - -k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_M \geqq r$ $\delta_{EC} \geqq r$ $\delta_{JC} \geqq r$	$k_b = 1$ $\delta_M \geqq r$ $\delta_{EC} \geqq r$ $\delta_{JC} \geqq r$	$\nexists$	$\delta_M < r \vee$ $\vee \delta_{EC} < r \vee$ $\vee \delta_{JC} < r$
BG8	$rs_bK + \frac{rs_bK}{\min\{r,\delta_M\}} + + \frac{rs_bK}{\min\{r,\delta_M,\delta_{JC}\}} + + \frac{r(k_b-1)s_bK}{\min\{r,\delta_M,\delta_{JC}\}} - -k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_M \geqq r$ $\delta_{JC} \geqq r$	$k_b = 1$ $\delta_M \geqq r$ $\delta_{JC} \geqq r$	$\nexists$	$\delta_M < r \vee$ $\vee \delta_{JC} < r$
BG9	$rs_bK + \frac{rs_bK}{\min\{r,\delta_M\}} + + \frac{rs_bK}{\min\{r,\delta_M,\delta_{EC}\}} + + \frac{rs_bK}{\min\{r,\delta_{EC},\delta_{JC}\}} + + \frac{r(k_b-1)s_bK}{\min\{r,\delta_M,\delta_{EC},\delta_{JC}\}} - -k_bs_bK$	$\nexists$	$k_b > 1$ $\delta_M \geqq r$ $\delta_{EC} \geqq r$ $\delta_{JC} \geqq r$	$k_b = 1$ $\delta_M \geqq r$ $\delta_{EC} \geqq r$ $\delta_{JC} \geqq r$	$\nexists$	$\delta_M < r \vee$ $\vee \delta_{EC} < r \vee$ $\vee \delta_{JC} < r$

Table 5.3: Overhead and scalability of WESW bulk data staging

Group	Latency ($\Theta_b(a)$)	$O(0)$	$O(1)$	$O(l_b)$	$O(r)$	$O(rl_b)$
BG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG2	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
BG3	s_bK	$\nexists$	$k_b > 1$	$k_b = 1$	$k_b > 1$	$k_b = 1$
BG4	$\frac{rs_bK}{\min\{r,\delta_M\}}$	$\nexists$	$k_b > 1$ $\delta_M \geqq r$	$k_b = 1$ $\delta_M \geqq r$	$k_b > 1 \vee$ $\vee \delta_M < r$	$k_b = 1 \vee$ $\vee \delta_M < r$
BG5	$\frac{rs_bK}{\min\{r,\delta_{EC}\}}$	$\nexists$	$k_b > 1$ $\delta_{EC} \geqq r$	$k_b = 1$ $\delta_{EC} \geqq r$	$k_b > 1 \vee$ $\vee \delta_{EC} < r$	$k_b = 1 \vee$ $\vee \delta_{EC} < r$
BG6	$s_bK + \frac{rs_bK}{\min\{r,\delta_{EC}\}}$	$\nexists$	$k_b > 1$ $\delta_{EC} \geqq r$	$k_b = 1$ $\delta_{EC} \geqq r$	$k_b > 1 \vee$ $\vee \delta_{EC} < r$	$k_b = 1 \vee$ $\vee \delta_{EC} < r$
BG7	$\frac{rs_bK}{\min\{r,\delta_M\}} + \frac{rs_bK}{\min\{r,\delta_M,\delta_{EC}\}}$	$\nexists$	$k_b > 1$ $\delta_M \geqq r$ $\delta_{EC} \geqq r$	$k_b = 1$ $\delta_M \geqq r$ $\delta_{EC} \geqq r$	$k_b > 1 \vee$ $\vee \delta_M < r \vee$ $\vee \delta_{EC} < r$	$k_b = 1 \vee$ $\vee \delta_M < r \vee$ $\vee \delta_{EC} < r$
BG8	$s_bK + \frac{rs_bK}{\min\{r,\delta_M\}}$	$\nexists$	$k_b > 1$ $\delta_M \geqq r$	$k_b = 1$ $\delta_M \geqq r$	$k_b > 1 \vee$ $\vee \delta_M < r$	$k_b = 1 \vee$ $\vee \delta_M < r$
BG9	$s_bK + \frac{rs_bK}{\min\{r,\delta_M\}} +$ $+ \frac{rs_bK}{\min\{r,\delta_M,\delta_{EC}\}} +$	$\nexists$	$k_b > 1$ $\delta_M \geqq r$ $\delta_{EC} \geqq r$	$k_b = 1$ $\delta_M \geqq r$ $\delta_{EC} \geqq r$	$k_b > 1 \vee$ $\vee \delta_M < r \vee$ $\vee \delta_{EC} < r$	$k_b = 1 \vee$ $\vee \delta_M < r \vee$ $\vee \delta_{EC} < r$

Table 5.4: Latency and scalability of WESW bulk data staging

data staging, cases of group BG2 always provide 0 latency, but overhead is 0 if and only if $\delta_{JC} \geqq r$, otherwise overhead is linear with both l_b and r. Cases of group BG3 – BG9 never provide 0 latency nor overhead and the same rules apply as in the case of DASG bulk data transfer groups BG2 and BG3 described in section 2.3.3.

Case	Bulk dara path	Instance layout	Restrictions	Group
BC1 *	(JP, JC)	$(JP, JC) \in h$	$\delta_{JC} = r$	BG1
BC2	(JP, EC, JC)	$(JP, EC), (EC, JC) \in h$	$\delta_{EC}, \delta_{JC} = r$	
BC3	(JP, EP, JC)	$(JP, EP), (EP, JC) \in h$	$\delta_{JC} = r$	
BC4	(JP, M, JC)	$(JP, M), (M, JC) \in h$	$\delta_{M}, \delta_{JC} = r$	
BC5	(JP, EP, EC, JC)	$(JP, EP), (EP, EC),$ $(EC, JC) \in h$	$\delta_{EC}, \delta_{JC} = r$	
BC6	(JP, M, EC, JC)	$(JP, M), (M, EC),$ $(EC, JC) \in h$	$\delta_{M}, \delta_{EC}, \delta_{JC} = r$	
BC7	(JP, EP, M, JC)	$(JP, EP), (EP, M),$ $(M, JC) \in h$	$\delta_{M}, \delta_{JC} = r$	
BC8	(JP, EP, M, EC, JC)	$(JP, EP), (EP, M),$ $(M, EC), (EC, JC) \in h$	$\delta_{M}, \delta_{EC}, \delta_{JC} = r$	
BC9 *	(JP, JC)	$(JP, JC) \notin h$		BG2
BC10	(JP, EC, JC)	$(JP, EC) \in h$ $(EC, JC) \notin h$	$\delta_{EC} = r$	
BC11	(JP, EC, JC)	$(JP, EC) \notin h$ $(EC, JC) \in h$	$\delta_{EC} = \delta_{JC}$	
BC12	(JP, EP, JC)	$(JP, EP) \in h$ $(EP, JC) \notin h$		
BC13	(JP, EP, JC)	$(JP, EP) \notin h$ $(EP, JC) \in h$	$\delta_{JC} = r$	
BC14	(JP, M, JC)	$(JP, M) \in h$ $(M, JC) \notin h$	$\delta_{M} = r$	
BC15	(JP, M, JC)	$(JP, M) \notin h$ $(M, JC) \in h$	$\delta_{M} = \delta_{JC}$	
BC16	(JP, EP, EC, JC)	$(JP, EP), (EP, EC) \in h$ $(EC, JC) \notin h$	$\delta_{EC} = r$	
BC17	(JP, EP, EC, JC)	$(JP, EP), (EC, JC) \in h$ $(EP, EC) \notin h$	$\delta_{EC} = \delta_{JC}$	
BC18	(JP, EP, EC, JC)	$(EP, EC), (EC, JC) \in h$ $(JP, EP) \notin h$	$\delta_{EC}, \delta_{JC} = r$	
BC19	(JP, M, EC, JC)	$(JP, M), (M, EC) \in h$ $(EC, JC) \notin h$	$\delta_{M}, \delta_{EC} = r$	
BC20	(JP, M, EC, JC)	$(JP, M), (EC, JC) \in h$ $(M, EC) \notin h$	$\delta_{M} = r$ $\delta_{EC} = \delta_{JC}$	

Table 5.5: WESW bulk data flow cases part 1/3.

Case	Bulk dara path	Instance layout	Restrictions	Group
BC21	(JP, M, EC, JC)	$(M, EC), (EC, JC) \in h$ $(JP, M) \notin h$	$\delta_M = \delta_{EC} = \delta_{JC}$	BG2
BC22	(JP, EP, M, JC)	$(JP, EP), (EP, M) \in h$ $(M, JC) \notin h$	$\delta_M = r$	
BC23	(JP, EP, M, JC)	$(JP, EP), (M, JC) \in h$ $(EP, M) \notin h$	$\delta_M = \delta_{JC}$	
BC24	(JP, EP, M, JC)	$(EP, M), (M, JC) \in h$ $(JP, EP) \notin h$	$\delta_M, \delta_{JC} = r$	
BC25	(JP, EP, M, EC, JC)	$(JP, EP), (EP, M),$ $(M, EC) \in h, (EC, JC) \notin h$	$\delta_M, \delta_{EC} = r$	
BC26	(JP, EP, M, EC, JC)	$(JP, EP)(EP, M),$ $(EC, JC) \in h, (M, EC) \notin h$	$\delta_M = r$ $\delta_{EC} = \delta_{JC}$	
BC27	(JP, EP, M, EC, JC)	$(JP, EP), (EC, JC),$ $(M, EC) \in h, (EP, M) \notin h$	$\delta_M = \delta_{EC} = \delta_{JC}$	
BC28	(JP, EP, M, EC, JC)	$(EC, JC), (EP, M),$ $(M, EC) \in h, (JP, EP) \notin h$	$\delta_{JC}, \delta_M, \delta_{EC} = r$	
BC29 *	(JP, EP, JC)	$(JP, EP), (EP, JC) \notin h$		BG3
BC30	(JP, EP, EC, JC)	$(JP, EP), (EP, EC) \notin h$ $(EC, JC) \in h$	$\delta_{EC} = \delta_{JC}$	
BC31	(JP, EP, M, JC)	$(JP, EP), (EP, M) \notin h$ $(M, JC) \in h$	$\delta_M = \delta_{JC}$	
BC32	(JP, EP, M, EC, JC)	$(JP, EP)(EP, M) \notin h$ $(EC, JC), (M, EC) \in h$	$\delta_{EC} = \delta_{JC} = \delta_M$	
BC33 *	(JP, M, JC)	$(JP, M), (M, JC) \notin h$		BG4
BC34	(JP, M, EC, JC)	$(JP, M), (M, EC) \notin h$ $(EC, JC) \in h$	$\delta_{EC} = \delta_{JC}$	
BC35	(JP, EP, M, JC)	$(EP, M), (M, JC) \notin h$ $(JP, EP) \in h$		
BC36	(JP, EP, M, EC, JC)	$(EP, M), (M, EC) \notin h$ $(JP, EP), (EC, JC) \in h$	$\delta_{EC} = \delta_{JC}$	
BC37 *	(JP, EC, JC)	$(JP, EC), (EC, JC) \notin h$		BG5
BC38	(JP, EP, EC, JC)	$(EP, EC), (EC, JC) \notin h$ $(JP, EP) \in h$		
BC39	(JP, M, EC, JC)	$(M, EC), (EC, JC) \notin h$ $(JP, M) \in h$	$\delta_M = r$	
BC40	(JP, EP, M, EC, JC)	$(M, EC), (EC, JC) \notin h$ $(JP, EP), (EP, M) \in h$	$\delta_M = r$	

Table 5.6: WESW bulk data flow cases part 2/3.

Case	Bulk dara path	Instance layout	Restrictions	Group
BC41	(JP, EP, M, JC)	$(JP, EP), (M, JC) \notin h$ $(EP, M) \in h$	$\delta_M = r$	BG3 $\wedge$ BG4
BC42	(JP, EP, M, EC, JC)	$(JP, EP)(M, EC) \notin h$ $(EP, M), (EC, JC) \in h$	$\delta_M = r$ $\delta_{EC} = \delta_{JC}$	
BC43	(JP, EP, EC, JC)	$(JP, EP), (EC, JC) \notin h$ $(EP, EC) \in h$	$\delta_{EC} = r$	BG3 $\wedge$ BG5
BC44	(JP, M, EC, JC)	$(JP, M), (EC, JC) \notin h$ $(M, EC) \in h$	$\delta_M = \delta_{EC}$	BG4 $\wedge$ BG5
BC45	(JP, EP, M, EC, JC)	$(EP, M), (EC, JC) \notin h$ $(JP, EP), (M, EC) \in h$	$\delta_{EC} = \delta_M$	
BC46	(JP, EP, M, EC, JC)	$(JP, EP), (EC, JC) \notin h$ $(M, EC), (EP, M) \in h$	$\delta_M, \delta_{EC} = r$	BG3 $\wedge$ BG4 $\wedge$ BG5
BC47 *	(JP, EP, EC, JC)	$(JP, EP), (EP, EC),$ $(EC, JC) \notin h$		BG6
BC48 *	(JP, M, EC, JC)	$(JP, M), (M, EC),$ $(EC, JC) \notin h$		BG7
BC49	(JP, EP, M, EC, JC)	$(EC, JC), (EP, M),$ $(M, EC) \notin h, (JP, EP) \in h$		
BC50 *	(JP, EP, M, JC)	$(JP, EP), (EP, M),$ $(M, JC) \notin h$		BG8
BC51	(JP, EP, M, EC, JC)	$(JP, EP)(EP, M),$ $(M, EC) \notin h, (EC, JC) \in h$	$\delta_{EC} = \delta_{JC}$	
BC52	(JP, EP, M, EC, JC)	$(JP, EP), (EC, JC),$ $(M, EC) \notin h, (EP, M) \in h$	$\delta_M = r$	BG6 $\wedge$ BG7
BC53	(JP, EP, M, EC, JC)	$(JP, EP)(EP, M),$ $(EC, JC) \notin h, (M, EC) \in h$	$\delta_M = \delta_{EC}$	BG6 $\wedge$ BG8
BC54 *	(JP, EP, M, EC, JC)	$(JP, EP), (EP, M),$ $(M, EC), (EC, JC) \notin h$		BG9

Table 5.7: WESW bulk data flow cases part 3/3.

5.3.4 WESW engine and workflow flow

Case	Engine path	Instance layout	Restrictions	Group
EC1 *	(RE, EC)	$(RE, EC) \in h$	$\delta_{RE} = \delta_{EC}$	EG1
EC2	(RE, M, EC)	$(RE, M), (M, EC) \in h$	$\delta_{RE} = \delta_M = \delta_{EC}$	
EC3	(RE, EP, EC)	$(RE, EP), (EP, EC) \in h$	$\delta_{RE}, \delta_{EC} = r$	
EC4	(RE, M, EP, EC)	$(RE, M), (M, EP), (EP, EC) \in h$	$\delta_{RE}, \delta_M, \delta_{EC} = r$	
EC5 *	(RE, EC)	$(RE, EC) \notin h$		EG2
EC6	(RE, M, EC)	$(RE, M) \in h \wedge (M, EC) \notin h$	$\delta_M = \delta_{RE}$	
EC7	(RE, M, EC)	$(RE, M) \notin h \wedge (M, EC) \in h$	$\delta_M = \delta_{EC}$	
EC8	(RE, EP, EC)	$(RE, EP) \in h \wedge (EP, EC) \notin h$	$\delta_{RE} = r$	
EC9	(RE, EP, EC)	$(RE, EP) \notin h \wedge (EP, EC) \in h$	$\delta_{EC} = r$	
EC10	(RE, M, EP, EC)	$(RE, M), (M, EP) \in h \wedge$ $\wedge (EP, EC) \notin h$	$\delta_M, \delta_{RE} = r$	
EC11	(RE, M, EP, EC)	$(RE, M), (EP, EC) \in h \wedge$ $\wedge (M, EP) \notin h$	$\delta_{EC} = r$ $\delta_M = \delta_{RE}$	
EC12	(RE, M, EP, EC)	$(M, EP), (EP, EC) \in h \wedge$ $\wedge (RE, M) \notin h$	$\delta_M, \delta_{EC} = r$	
EC13 *	(RE, M, EC)	$(RE, M), (M, EC) \notin h$		EG3
EC14	(RE, M, EP, EC)	$(EP, EC) \in h \wedge$ $\wedge (RE, M), (M, EP) \notin h$	$\delta_{EC} = r$	
EC15	(RE, M, EP, EC)	$(M, EP) \in h \wedge$ $\wedge (RE, M), (EP, EC) \notin h$	$\delta_M = r$	EG3 $\wedge$ EG4
EC16 *	(RE, EP, EC)	$(RE, EP), (EP, EC) \notin h$		EG4
EC17	(RE, M, EP, EC)	$(RE, M) \in h \wedge$ $\wedge (M, EP), (EP, EC) \notin h$	$\delta_{RE} = \delta_M$	
EC18 *	(RE, M, EP, EC)	$(RE, M), (M, EP), (EP, EC) \notin h$		EG5

Table 5.8: WESW engine data flow cases

Engine and workflow flow analysis is similar to bulk data flow analysis in several aspects. Based definition 2.54, the four engine and four workflow path layout types (see definition 5.8 and 5.9), 18 engine and 18 workflow data flow cases can be identified. These are listed in table 5.8 and table 5.9. Restrictions implied by instance layout are also included in the tables and representative cases (see definition 2.54) are marked with asterisks.

In terms of both engine and workflow transfer, the different cases can be divided

Case	Workflow path	Instance layout	Restrictions	Group
WC1 *	(RW, EC)	$(RW, EC) \in h$	$\delta_{RW} = \delta_{EC}$	WG1
WC2	(RW, M, EC)	$(RW, M), (M, EC) \in h$	$\delta_{RW} = \delta_M = \delta_{EC}$	
WC3	(RW, EP, EC)	$(RW, EP), (EP, EC) \in h$	$\delta_{RW}, \delta_{EC} = r$	
WC4	(RW, M, EP, EC)	$(RW, M), (M, EP), (EP, EC) \in h$	$\delta_{RW}, \delta_M, \delta_{EC} = r$	
WC5 *	(RW, EC)	$(RW, EC) \notin h$		WG2
WC6	(RW, M, EC)	$(RW, M) \in h \wedge (M, EC) \notin h$	$\delta_{RW} = \delta_M$	
WC7	(RW, M, EC)	$(RW, M) \notin h \wedge (M, EC) \in h$	$\delta_M = \delta_{EC}$	
WC8	(RW, EP, EC)	$(RW, EP) \in h \wedge (EP, EC) \notin h$	$\delta_{RW} = r$	
WC9	(RW, EP, EC)	$(RW, EP) \notin h \wedge (EP, EC) \in h$	$\delta_{EC} = r$	
WC10	(RW, M, EP, EC)	$(RW, M), (M, EP) \in h \wedge$ $\wedge (EP, EC) \notin h$	$\delta_{RW}, \delta_M = r$	
WC11	(RW, M, EP, EC)	$(RW, M), (EP, EC) \in h \wedge$ $\wedge (M, EP) \notin h$	$\delta_{EC} = r$ $\delta_{RW} = \delta_M$	
WC12	(RW, M, EP, EC)	$(M, EP), (EP, EC) \in h \wedge$ $\wedge (RW, M) \notin h$	$\delta_M, \delta_{EC} = r$	
WC13 *	(RW, M, EC)	$(RW, M), (M, EC) \notin h$		WG3
WC14	(RW, M, EP, EC)	$(EP, EC) \in h \wedge$ $\wedge (RW, M), (M, EP) \notin h$	$\delta_{EC} = r$	
WC15	(RW, M, EP, EC)	$(M, EP) \in h \wedge$ $\wedge (RW, M), (EP, EC) \notin h$	$\delta_M = r$	WG3 $\wedge$ WG4
WC16 *	(RW, EP, EC)	$(RW, EP), (EP, EC) \notin h$		WG4
WC17	(RW, M, EP, EC)	$(RW, M) \in h \wedge$ $\wedge (M, EP), (EP, EC) \notin h$	$\delta_{RW} = \delta_M$	
WC18 *	(RW, M, EP, EC)	$(RW, M), (M, EP), (EP, EC) \notin h$		WG5

Table 5.9: WESW workflow data flow cases

into 5 groups based on definition 2.55. Performance properties are the same in each case of the same group. Transfer time values are determined based on the definition of $\Gamma_e(a)$ and $\Gamma_w(a)$, and are included in table 5.10 and 5.11.

Cases of group EG1 and WG1 require, that source and destination nodes are hosted by the same machine, which implies that transfer time is 0. In cases of group EG2, EG3, EG4, EG5, WG2, WG3, WG4 and WG5, the same rules apply as in the case of WESA engine and workflow transfer groups EG2, EG3, WG2, and WG3 described in section 4.3.4. Scalability values are listed in table 5.10 and 5.11.

Group	Transfer time $\Gamma_e(a))$	$O(0)$	$O(1)$	$O(l_e)$	$O(r)$	$O(rl_e)$
EG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
EG2	$\frac{rk_e s_e K}{\min\{r,\delta_{EC},\delta_{RE}\}}$	$\nexists$	$\nexists$	$\delta_{RE} \geqq r$ $\delta_{EC} \geqq r$	$\nexists$	$\delta_{RE} < r \vee \delta_{EC} < r$
EG3	$\frac{rs_e K}{\min\{r,\delta_{RE},\delta_M\}} +$ $+ \frac{rs_e K}{\min\{r,\delta_M,\delta_{EC}\}} +$ $+ \frac{r(k_e-1)s_e K}{\min\{r,\delta_M,\delta_{RE},\delta_{EC}\}}$	$\nexists$	$\nexists$	$\delta_M \geqq r$ $\delta_{EC} \geqq r$ $\delta_{RE} \geqq r$	$\nexists$	$\delta_M < r \vee \delta_{RE} < r \vee \delta_{EC} < r$
EG4	$\frac{rs_e K}{\min\{r,\delta_{RE}\}} +$ $+ \frac{rs_e K}{\min\{r,\delta_{EC}\}} +$ $+ \frac{r(k_e-1)s_e K}{\min\{r,\delta_{RE},\delta_{EC}\}}$	$\nexists$	$\nexists$	$\delta_{EC} \geqq r$ $\delta_{RE} \geqq r$	$\nexists$	$\delta_{RE} < r \vee \delta_{EC} < r$
EG5	$\frac{rs_e K}{\min\{r,\delta_{RE},\delta_M\}} +$ $+ \frac{rs_e K}{\min\{r,\delta_M\}} +$ $+ \frac{rs_e K}{\min\{r,\delta_{EC}\}} +$ $+ \frac{r(k_e-1)s_e K}{\min\{r,\delta_M,\delta_{RE},\delta_{EC}\}}$	$\nexists$	$\nexists$	$\delta_M \geqq r$ $\delta_{EC} \geqq r$ $\delta_{RE} \geqq r$	$\nexists$	$\delta_M < r \vee \delta_{RE} < r \vee \delta_{EC} < r$

Table 5.10: Time and scalability of WESW engine transfer

Group	Transfer time $\Gamma_w(a))$	$O(0)$	$O(1)$	$O(l_w)$	$O(r)$	$O(rl_w)$
WG1	0	$\forall$	$\nexists$	$\nexists$	$\nexists$	$\nexists$
WG2	$\frac{rk_w s_w K}{\min\{r,\delta_{EC},\delta_{RW}\}}$	$\nexists$	$\nexists$	$\delta_{RW} \geqq r$ $\delta_{EC} \geqq r$	$\nexists$	$\delta_{RW} < r \vee \delta_{EC} < r$
WG3	$\frac{rs_w K}{\min\{r,\delta_{RW},\delta_M\}} +$ $+ \frac{rs_w K}{\min\{r,\delta_M,\delta_{EC}\}} +$ $+ \frac{r(k_w-1)s_w K}{\min\{r,\delta_M,\delta_{RW},\delta_{EC}\}}$	$\nexists$	$\nexists$	$\delta_M \geqq r$ $\delta_{EC} \geqq r$ $\delta_{RW} \geqq r$	$\nexists$	$\delta_M < r \vee \delta_{RW} < r \vee \delta_{EC} < r$
WG4	$\frac{rs_w K}{\min\{r,\delta_{RW}\}} +$ $+ \frac{rs_w K}{\min\{r,\delta_{EC}\}} +$ $+ \frac{r(k_w-1)s_w K}{\min\{r,\delta_{RW},\delta_{EC}\}}$	$\nexists$	$\nexists$	$\delta_{EC} \geqq r$ $\delta_{RW} \geqq r$	$\nexists$	$\delta_{RW} < r \vee \delta_{EC} < r$
WG5	$\frac{rs_w K}{\min\{r,\delta_{RW},\delta_M\}} +$ $+ \frac{rs_w K}{\min\{r,\delta_M\}} +$ $+ \frac{rs_w K}{\min\{r,\delta_{EC}\}} +$ $+ \frac{r(k_w-1)s_w K}{\min\{r,\delta_M,\delta_{RW},\delta_{EC}\}}$	$\nexists$	$\nexists$	$\delta_M \geqq r$ $\delta_{EC} \geqq r$ $\delta_{RW} \geqq r$	$\nexists$	$\delta_M < r \vee \delta_{RW} < r \vee \delta_{EC} < r$

Table 5.11: Time and scalability of WESW workflow transfer

5.3.5 Recommended WESW structure layout, data flow, and resource layout

Table 5.3 and 5.4 shows that in any of the bulk data transfer cases, it is possible to realise scalable bulk data transfer, where overhead and latency on bulk data transfer are independent of r and l_b. According to table 5.10 and 5.11, the same is true for engine and workflow staging, but not under the same conditions. The selection of proposed structure and data flow combinations is based on the following recommendations.

R1 - Mediator For the same reasons described in WESA recommendations R1 in section 4.3.5, it is aimed to minimise the number of utilised dedicated machines (let $\delta_M = 1$), $\delta_{JC} \geqq r$ should always be provided, and bulk data flow cases that transfer data via the mediator and data flow cases which require to have (M, EC) coupled are not recommended. Because in each scenario $\delta_{EP}, \delta_{JP} = r$, data flow cases which require to have (EP, M) or (JP, M) are not recommended. Furthermore, because of $\delta_{JC} \geqq r$, cases that require $(M, JC) \in h$ are not recommended either.

R2 - Child engine execution On the other hand, it cannot be guaranteed in general that parent and child workflow engines can be hosted by the same machine and it cannot be guaranteed that parent and child workflow jobs can be coupled with each other, or with each other's engine. Based on these, instance layouts having any of the following couplings are not recommended: (EP, EC), (JP, JC) (JP, EC), (EP, JC). This also implies that child engine is never recommended to be executed on external machines ($\xi_{EC} \neq ExM$).

R3 - Engine and workflow repositories There are workflow systems of which workflow engines are coupled with workflow repositories, for instance the P-GRADE

Recommendation	WESW data flow case
R1	BC4, BC6-BC8, BC14, BC15, BC19-BC28, BC31-BC36, BC39-BC42, BC44-BC46, BC48-BC54, EC2, EC4, EC7, EC10, EC12-EC14, EC18, WC2, WC4, WC7, WC10, WC12-WC14, WC18
R2	BC1-BC8, BC10, BC13, BC16, BC18, BC28, BC43, BC46, EC3, EC4, EC9, EC11, EC12, WC3, WC4, WC9, WC11, WC12
R3	EC3, EC8, EC10
R4	EC3, EC4, EC8 - EC12, EC14-EC18, WC3, WC4, WC8 - WC12, WC14-WC18

Table 5.12: Elimination of WESW data flow cases based on different recommendations.

portal which is coupled with a simplified workflow repository where users can store their private workflows. However, there are no workflow systems having their engines coupled with engine repository. Therefore, instance layouts which allow that parent workflow engines and workflow engine repositories are coupled are not considered, but instance layouts which allow that parent workflow engines and workflow repositories are coupled can be recommended. Based on this, instance layouts having (EP, RE) coupled are not recommended.

R4 - Engine and workflow transfer In special cases it might be preferred to transfer child engines and workflows via the parent workflow engine, in order to minimise bulk data transfer time, but in general it is preferred to transfer them directly from the repository machine to the machine where the child engine is executed, since transferring them via the parent workflow engine increases overall overhead and latency. Therefore, workflow and engine transfer cases that include the parent workflow engine in their path are not recommended.

Based on these recommendations, data flow cases listed in table 5.12 are excluded. Therefore, data transfer cases BC9, BC11, BC12, BC17, BC29, BC30, BC37, BC38, BC47, engine transfer cases EC1, EC5, EC6, and workflow transfer cases WC1, WC5, WC6 can be recommended. Table 5.13 illustrates all combinations

			Proposed																																			
Aspects			PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8a, PC8b	PC9a, PC9b	PC10a, PC10b	PC11a, PC11b	PC12a, PC12b	PC13a, PC13b	PC14a, PC14b	PC15a, PC15b	PC16a, PC16b	PC17a, PC17b	PC18a, PC18b	PC19a, PC19b	PC20a, PC20b	PC21a, PC21b	PC22a, PC22b	PC23a, PC23b	PC24a, PC24b	PC25a, PC25b	PC26a, PC26b	PC27a, PC27b	PC28a, PC28b	PC29	PC30	PC31	PC32	PC33a, PC33b	PC34a, PC34b	PC35a, PC35b	PC36a, PC36b
Structure	Instance layout	(EP,M)																																				
		(EP,RE)																																				
		(EP,RW)			X	X						X	X						X	X						X	X					X				X		
		(EP,EC)																																				
		(M,RE)		X		X			X		X		X			X		X		X			X		X		X			X								
		(M,RW)					X		X					X		X					X		X					X		X			X				X	
		(M,EC)																																				
		(RW,RE)						X	X						X	X						X	X						X	X				X				X
		(RW,EC)																																X				X
		(RE,EC)																													X	X	X	X	X	X	X	X
	Type layout	δ_M	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		δ_{RE}	≥1	1	≥1	1	≥1	≥1	1	≥1	1	≥1	1	≥1	≥1	1	≥1	1	≥1	1	≥1	≥1	1	≥1	1	≥1	1	≥1	≥1	1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1
		δ_{RW}	≥1	≥1	≥1	≥1	1	≥1	1	≥1	≥1	≥1	≥1	1	≥1	1	≥1	≥1	≥1	≥1	1	≥1	1	≥1	≥1	≥1	≥1	1	≥1	1	≥1	≥1	1	≥1	≥1	≥1	1	≥1
		δ_{EC}	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1
Data flow	Bulk data path	(JP,JC)	X	X	X	X	X	X	X																													
		(JP,EC,JC)	O*	O*	O*	O*	O*	O*	O*								X	X	X	X	X	X	X															
		(JP,EP,JC)	O*	O*	O*	O*	O*	O*	O*	X	X	X	X	X	X	X																						
		(JP,M,JC)																																				
		(JP,EP,EC,JC)	O*	O*	O*	O*	O*	O*	O*								O*	O*	O*	O*	O*	O*	O*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		(JP,M,EC,JC)																																				
		(JP,EP,M,JC)																																				
		(JP,EP,M,EC,JC)																																				
	Bulk data staging	Pipelined (k_b=1)	O	O	O	O	O	O	O	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	O	O	O	O	X^a	X^a	X^a	X^a
		Not pipelined (k_b>1)	O	O	O	O	O	O	O	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	O	O	O	O	X^b	X^b	X^b	X^b
	Engine path	(RE,EC)	X	O	X	O	X	X	O	X	O	X	O	X	X	O	X	O	X	O	X	X	O	X	O	X	O	X	X	O	X	X	X	X	X	X	X	X
		(RE,M,EC)		O		O			O		O		O			O		O		O			O		O		O			O								
		(RE,EP,EC)																																				
		(RE,M,EP,EC)																																				
	Engine staging	Pipelined (k_d>1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_d=1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
	Workflow path	(RW,EC)	X	X	X	X	O	X	O	X	X	X	X	O	X	O	X	X	X	X	O	X	O	X	X	X	X	O	X	O	X	O	O	X	X	O	O	X
		(RW,M,EC)					O		O					O		O					O		O					O		O			O				O	
		(RW,EP,EC)																														O				O		
		(RW,M,EP,EC)																																				
	Workflow staging	Pipelined (k_d>1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_d=1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Resource	Resource layout	Mediator	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
		Engine Repository	S	D	S	D	S	S	D	S	D	S	D	S	S	D	S	D	S	D	S	S	D	S	D	S	D	S	S	D	D	D	D	D	D	D	D	D
		Workflow Repository	S	S	E	E	S	S	S	S	S	E	E	S	S	S	S	S	E	E	S	S	S	S	S	E	E	S	S	S	S	E	S	S	S	E	S	S
		Engine execution	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	D	D	D	D	D	D	D	D

Table 5.13: Proposed WESW structures, data flows, and resource layouts, where sign X within instance layout show which nodes are coupled in a particular case. Sign X within data flow means that only the correlated architectural approach can be implemented, sign O indicates that it is irrelevant which approach is chosen, while sign O* means the same as O, but it only applies if instance layout defines node types EC and JC or EP and WE are coupled. Sign X^a and X^b mean that it is possible to realise both pipelined (X^a) and non pipelined (X^b) bulk data staging by the given structure, but this affects the performance characteristics as illustrated in table 5.15 and 5.16. Within resource layout D represents dedicated, C represents computational, S represents storage, and E represents external machines.

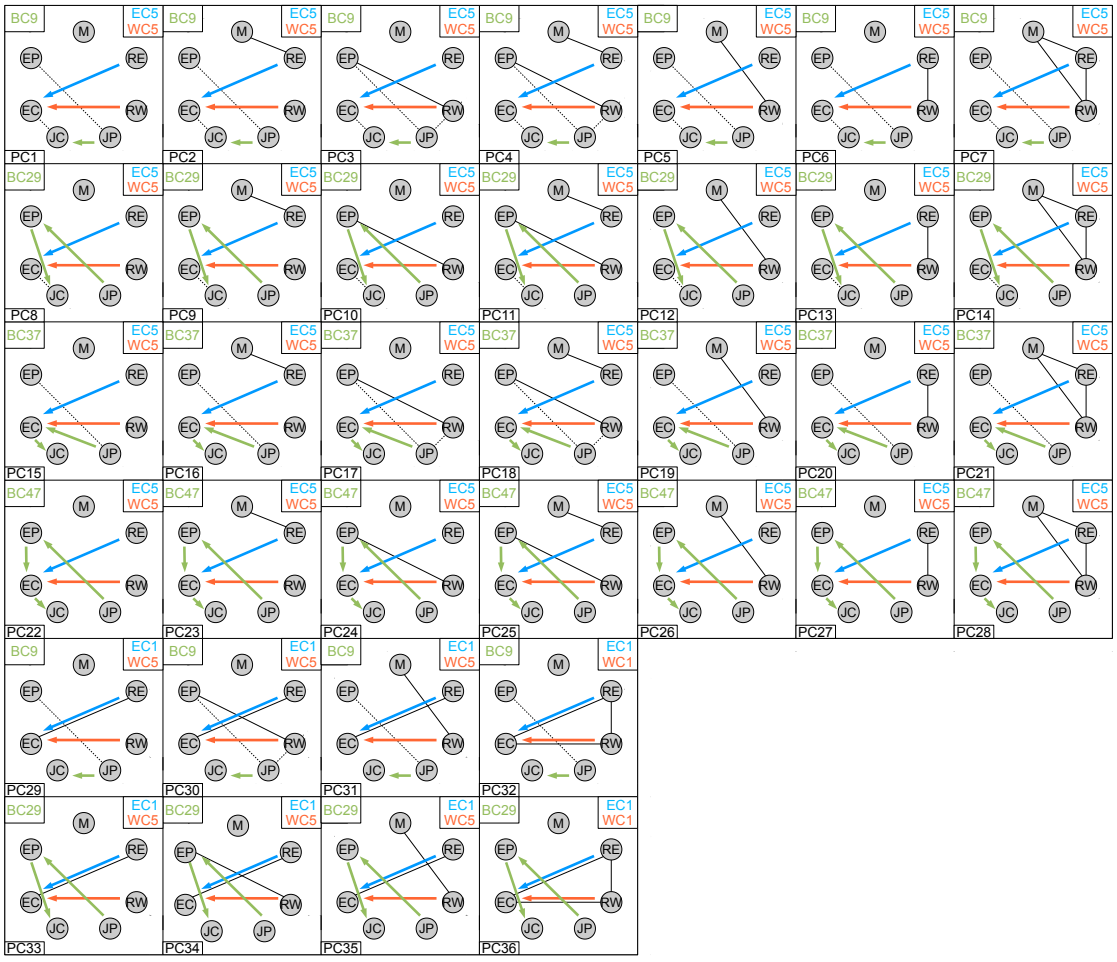

Figure 5.4: Combinations of recommended representative WESW data flow cases, where black lines represent which node types are coupled, dashed lines represent optional couplings, green arrows represent bulk data path layouts, blue arrows represent engine path layouts, and orange arrows represent workflow path layouts.

of these data flow cases and structure layouts that can implement them. Obviously, different data flow types can be recommended under different circumstances. Cases are classified based on the following aspects:

1. Parent workflow jobs

- **(a)** If parent workflow jobs are executed locally to the parent workflow engine ($(EP, JP) \in h$), then bulk data transfer BC29, BC30, and BC47 cannot be applied, since these explicitly require that $(EP, JP) \notin h$.

- **(b)** If parent workflow jobs are executed remotely to the parent workflow engine ($(EP, JP) \notin h$) then bulk data transfer BC12, BC17, BC38 cannot be applied, since these explicitly require that $(EP, JP) \in h$.

  (b-i) If this is the case and data can be transferred directly to/from the parent job machine from a third party computational resource, then BC9, BC11, BC37 can be recommended, since BC29, BC30, BC47 transfer data via the parent workflow engine that adds extra overhead and latency. (See performance properties of these transfer cases in section 5.3.3).

  (b-ii) If parent workflow jobs are executed remotely to the parent engine and data can only be transferred to/from the parent job machine via the parent engine, then BC29, BC30, or BC47 are recommended, since BC9, BC11, BC37 cannot be applied.

2. Child workflow jobs

- **(a)** Similarly to parent workflow jobs, if child workflow jobs are executed locally to the child workflow engine ($(EC, JC) \in h$), then bulk data transfer BC37, BC38, and BC47 cannot be applied, since these explicitly require that $(EC, JC) \notin h$. In these cases EC1 is not recommended, since to utilise a previously installed workflow engines that are hosted on dedicated machines may bottleneck data transfer, therefore in $\xi_{EC} = CoM$.

- **(b)** If child workflow jobs are executed remotely to their workflow engines ($(EC, JC) \notin h$) then bulk data transfer BC11, BC17, BC30 cannot be applied, since these explicitly require that $(EC, JC) \in h$.

- **(b-i)** If this is the case and data can be transferred directly to/from the child job machine from a third party computational resource, then BC9, BC12, BC29 can be recommended, because BC37, BC38, BC47 transfer data via the child workflow engine increasing overhead and latency on bulk data transfer. (See again performance properties of these transfer cases in section 5.3.3.) In these cases (BC9, BC12, BC29), it is recommended to use child engines previously installed on dedicated machines ($(RE, EC) \in h, \xi_{EC} = ExM$) and apply EC1, since in this case bulk data is transferred directly to the child jobs. Therefore, previously installed child engines will not bottleneck bulk data transfer.

- **(b-ii)** If child workflow jobs are executed remotely to the child engine and data can only be transferred to/from the child job machine via the child engine, then BC9, BC12, BC29 cannot be applied. Hence, BC37, BC38, BC47 are recommended. In this case child engines are recommended to be executed on computational machines ($\xi_{EC} = CoM$) and EC1 is not recommended due to the same reason as defined in case where child workflow jobs are executed locally (see case 2/a).

3. Child workflow engines

- **(a)** If all child workflow engines are relatively small (up to a few megabytes), the engine repository can be coupled with the mediator ($(M, RE) \in h, \xi_{RE} = DeM$) without adding significant overhead resulted by engine transfer. Therefore, in this case EC5 or EC6 can be recommended.

- **(b)** If child engines are relatively large (hundreds of megabytes), then it is recommended to use multiple engine repository nodes hosted on storage machines ($(M, RE) \notin h, \xi_{RE} = StM$). In this case only EC5 is recommended, since EC6 would bottleneck engine transfer.

In special cases (see case 2/b-i) it is recommended to use workflow engines hosted on dedicated machines where EC1 is recommended independently of engine size. In this case $(RE, EC) \in h, \xi_{RE} = DeM$.

4. Child workflow descriptors

- **(a)** Child workflow descriptors can be hosted in such repositories where $((JP, RW), (EP, RW) \notin h)$, in which case

  (a-i) if all child workflow descriptors are relatively small (up to a few megabytes, e.g. only simple XML descriptors that contain the graph structure and references to jobs), the workflow repository can be coupled with the mediator $((M, RW) \in h, \xi_{RW} = DeM)$ without adding significant overhead resulted by workflow transfer. In this case both WC5 and WC6 can be recommended. However,

  (a-ii) if they are relatively large (hundreds of megabytes, e.g. they also contain large workflow jobs) it is recommended to have multiple workflow repositories hosted on storage machines $((M, RW) \notin h, \xi_{RW} = StM)$.

- **(b)** The workflow repository can also be coupled with the parent workflow engine which is represented by $(EP, RW) \in h$ and $\xi_{RW} = ExM$.

For the latter two (a-ii and b) only WC5 is recommended, since WC6 would bottleneck workflow transfer in the case of (b) and it cannot be realised in the case of (a-ii). However in the case when engine repository and child engine execution are coupled and hosted on dedicated machines $(RE, EC) \in h$, then workflow repository nodes can also be hosted on these machines $(RW, EC) \in h, \xi_{RW} = DeM$, in order to avoid latency and overhead increase resulted by workflow transfer. In this case, WC1 is recommended independently of workflow size.

Cases	3/a, 4/a-i	3/a, 4/a-ii	3/a, 4/b	3/b, 4/a-i	3/b, 4/a-ii	3/b, 4/b
1/a, 2/a	PC7	PC2	PC4	PC5	PC1, PC6	PC3
1/a, 2/b-i	PC31	PC29, P32	PC30	PC31	PC29, P32	PC30
1/a, 2/b-ii	PC21	PC16	PC18	PC19	PC15, PC20	PC17
1/b-i, 2/a	PC7	PC2	PC4	PC5	PC1, PC6	PC3
1/b-i, 2/b-i	PC31	PC29, P32	PC30	PC31	PC29, P32	PC30
1/b-i, 2/b-ii	PC21	PC16	PC18	PC19	PC15, PC20	PC17
1/b-ii, 2/a	PC14	PC9	PC11	PC12	PC8, PC13	PC10
1/b-ii, 2/b-i	PC35	PC33, PC36	PC34	PC35	PC33, PC36	PC34
1/b-ii, 2/b-ii	PC28	PC23	PC25	PC26	PC22, PC27	PC24

Table 5.14: Proposed WESW structure and data flow combinations in different cases

Table 5.14, summarizes which proposed structure and data flow combinations are recommended in the different cases. Table 5.15 and 5.16 illustrate performance characteristics of each proposed combination. BC9, BC11, BC12, BC17 all belong to group BG2; BC29, BC30 belong to BG3; BC37, BC38 belong to BG5; BC47 belongs to BG6; EC1 belongs to EG1; EC5, EC6 belong to EG2; WC1 belongs to WG1; and WC5, WC6 belong to WG2. Combinations of representative cases are shown in figure 5.4.

Proposed	Overhead / Latency	Overhead / Latency scalability
PC1, PC6	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + \frac{rl_wK}{\min\{r,\delta_{RW}\}}$	$O(rl_e + rl_w + 1)$
PC2	$w_e + rl_eK + \frac{rl_wK}{\min\{r,\delta_{RW}\}}$	$O(rl_e + rl_w + 1)$
PC3	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + l_wK$	$O(rl_e + l_w + 1)$
PC4	$w_e + rl_eK + l_wK$	$O(rl_e + l_w + 1)$
PC5	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + rl_wK$	$O(rl_e + rl_w + 1)$
PC7	$w_e + rl_eK + rl_wK$	$O(rl_e + rl_w + 1)$
PC8a, PC13a, PC15a, PC20a	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + s_bK$	$O(rl_e + rl_w + 1)$
PC8b, PC13b, PC15b, PC20b	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC9a, PC16a	$w_e + rl_eK + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + s_bK$	$O(rl_e + rl_w + 1)$
PC9b, PC16b	$w_e + rl_eK + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC10a, PC17a	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + l_wK + s_bK$	$O(rl_e + l_w + 1)$
PC10b, PC17b	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + l_wK + l_bK$	$O(rl_e + l_w + l_b + 1)$
PC11a, PC18a	$w_e + rl_eK + l_wK + s_bK$	$O(rl_e + l_w + 1)$
PC11b, PC18b	$w_e + rl_eK + l_wK + l_bK$	$O(rl_e + l_w + l_b + 1)$
PC12a, PC19a	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + rl_wK + s_bK$	$O(rl_e + rl_w + 1)$
PC12b, PC19b	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + rl_wK + l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC14a, PC21a	$w_e + rl_eK + rl_wK + s_bK$	$O(rl_e + l_w + 1)$
PC14b, PC21b	$w_e + rl_eK + rl_wK + l_bK$	$O(rl_e + l_w + l_b + 1)$
PC22a, PC27a	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + 2s_bK$	$O(rl_e + rl_w + 1)$
PC22b, PC27b	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + 2l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC23a	$w_e + rl_eK + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + 2s_bK$	$O(rl_e + rl_w + 1)$
PC23b	$w_e + rl_eK + \frac{rl_wK}{\min\{r,\delta_{RW}\}} + 2l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC24a	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + l_wK + 2s_bK$	$O(rl_e + l_w + 1)$
PC24b	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + l_wK + 2l_bK$	$O(rl_e + l_w + l_b + 1)$
PC25a	$w_e + rl_eK + l_wK + 2s_bK$	$O(rl_e + l_w + 1)$
PC25b	$w_e + rl_eK + l_wK + 2l_bK$	$O(rl_e + l_w + l_b + 1)$
PC26a	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + rl_wK + 2s_bK$	$O(rl_e + rl_w + 1)$
PC26b	$w_e + \frac{rl_eK}{\min\{r,\delta_{RE}\}} + rl_wK + 2l_bK$	$O(rl_e + rl_w + l_b + 1)$
PC28a	$w_e + rl_eK + rl_wK + 2s_bK$	$O(rl_e + l_w + 1)$
PC28b	$w_e + rl_eK + rl_wK + 2l_bK$	$O(rl_e + l_w + lb)$
PC29	$\frac{rl_wK}{\min\{r,\delta_{EC},\delta_{RW}\}}$	$O(rl_w)$
PC30	l_wK	$O(l_w)$

Table 5.15: Performance characteristics of proposed WESW structure and data data flow combinations part 1/2.

Proposed	Overhead / Latency	Overhead / Latency scalability
PC31	rl_wK	$O(rl_w)$
PC32	0	$O(0)$
PC33a	$\frac{rl_wK}{\min\{r,\delta_{EC},\delta_{RW}\}} + s_bK$	$O(rl_w + 1)$
PC33b	$\frac{rl_wK}{\min\{r,\delta_{EC},\delta_{RW}\}} + l_bK$	$O(rl_w + l_b)$
PC34a	$l_wK + s_bK$	$O(l_w + 1)$
PC34b	$l_wK + l_bK$	$O(l_w + l_b)$
PC35a	$rl_wK + s_bK$	$O(rl_w + 1)$
PC35b	$rl_wK + l_bK$	$O(rl_w + l_b)$
PC36a	s_bK	$O(1)$
PC36b	l_bK	$O(l_b)$

Table 5.16: Performance characteristics of proposed WESW structure and data data flow combinations part 2/2.

5.4 Existing and proposed WESW solutions

5.4.1 Existing WESW solutions

The Gria based solution that was already introduced in section 4.4 developed within the SIMDAT project, was connected to Taverna and InforSense KDE workflow systems in order to support the execution of particular workflows of different kinds from those systems. In theory, the Gria based approach could be used as a WESW. Another solution called VLE-WFBus [74, 115, 116] was developed at the Dutch Virtual Laboratory for e-Science. This solution connects a few popular workflow engines in order to create a meta-workflow system that allows the composition and execution of high-level heterogeneous workflows via the Vergil GUI.

The Gria based solution exposes the functionality of workflow engines via general frontend Web/Grid services that are invoked by the parent workflow engine.

Aspects			Existing solutions											
			Gria v1	Gria v2a,v2b	Gria v3a,v3b	Gria v4a,v4b	Gria v5	Gria v6	WFBus v1	WFBus v2a,v2b	WFBus v3a,v3b	WFBus v4a,v4b	WFBus v5	WFBus v6
Structure	Instance layout	(EP,M)												
		(EP,RE)												
		(EP,RW)	O	O	O	O	O	O	O	O	O	O	O	O
		(EP,EC)												
		(M,RE)												
		(M,RW)												
		(M,EC)												
		(RW,RE)												
		(RW,EC)												
		(RE,EC)	X	X	X	X	X	X	X	X	X	X	X	X
	Type layout	δ_M	1	1	1	1	1	1	1	1	1	1	1	1
		δ_{RE}	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1
		δ_{RW}	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1
		δ_{EC}	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1
Data flow	Bulk data path	(JP,JC)	X						X					
		(JP,EC,JC)	O*	X					O*	X				
		(JP,EP,JC)	O*		X				O*		X			
		(JP,M,JC)					O*						O*	
		(JP,EP,EC,JC)	O*	O*	O*	X			O*	O*	O*	X		
		(JP,M,EC,JC)					X						X	
		(JP,EP,M,JC)						O*						O*
		(JP,EP,M,EC,JC)						X						X
	Bulk data staging	Pipelined (k_b=1)	O	X^a	X^a	X^a			O	X^a	X^a	X^a		
		Not pipelined (k_b>1)	O	X^b	X^b	X^b	X	X	O	X^b	X^b	X^b	X	X
	Engine path	(RE,EC)	X	X	X	X	X	X	X	X	X	X	X	X
		(RE,M,EC)												
		(RE,EP,EC)												
		(RE,M,EP,EC)												
	Engine staging	Pipelined (k_d>1)	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_d=1)	O	O	O	O	O	O	O	O	O	O	O	O
	Workflow path	(RW,EC)	X	X	X	X	X	X	X	X	X	X	X	X
		(RW,M,EC)	O*	O*	O*	O*	O*	O*	O*	O*	O*	O*	O*	O*
		(RW,EP,EC)												
		(RW,M,EP,EC)												
	Workflow staging	Pipelined (k_d>1)	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_d=1)	O	O	O	O	O	O	O	O	O	O	O	O
Resource	Resource layout	Mediator	D	D	D	D	D	D	D	D	D	D	D	D
		Engine Repository	D	D	D	D	D	D	D	D	D	D	D	D
		Workflow Repository	E/D	E/D	E/D	E/D	E/D	E/D	E/D	E/D	E/D	E/D	E/D	E/D
		Engine execution	D	D	D	D	D	D	D	D	D	D	D	D
Integr.	Subject of integration	Auxiliary tool level												
		Workflow engine level	X	X	X	X	X	X	X	X	X	X	X	X
Interface	Backend	CLI							O	O	O	O	O	O
		API	X	X	X	X	X	X	O	O	O	O	O	O
	Frontend	General	X	X	X	X	X	X	X	X	X	X	X	X
		Specific												

Aspects			Proposed architectures																	
			PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8a, PC8b	PC9a, PC9b	PC10a, PC10b	PC11a, PC11b	PC12a, PC12b	PC13a, PC13b	PC14a, PC14b	PC15a, PC15b	PC16a, PC16b	PC17a, PC17b	PC18a, PC18b
Structure	Instance layout	(EP,M)																		
		(EP,RE)																		
		(EP,RW)			X	X						X	X						X	X
		(EP,EC)																		
		(M,RE)		X		X			X		X		X			X		X		X
		(M,RW)					X		X					X		X				
		(M,EC)																		
		(RW,RE)						X	X						X	X				
		(RW,EC)																		
		(RE,EC)																		
	Type layout	δ_M	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		δ_{RE}	≥1	1	≥1	1	≥1	≥1	1	≥1	1	≥1	1	≥1	≥1	1	≥1	1	≥1	1
		δ_{RW}	≥1	≥1	≥1	≥1	1	≥1	1	≥1	≥1	≥1	≥1	1	≥1	1	≥1	≥1	≥1	≥1
		δ_{EC}	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1
Data flow	Bulk data path	(JP,JC)	X	X	X	X	X	X	X											
		(JP,EC,JC)	O*	O*	O*	O*	O*	O*	O*								X	X	X	X
		(JP,EP,JC)	O*	O*	O*	O*	O*	O*	O*	X	X	X	X	X	X	X				
		(JP,M,JC)																		
		(JP,EP,EC,JC)	O*	O*	O*	O*	O*	O*	O*								O*	O*	O*	O*
		(JP,M,EC,JC)																		
		(JP,EP,M,JC)																		
		(JP,EP,M,EC,JC)																		
	Bulk data staging	Pipelined (k_b=1)	O	O	O	O	O	O	O	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a
		Not pipelined (k_b>1)	O	O	O	O	O	O	O	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b
	Engine path	(RE,EC)	X	O	X	O	X	X	O	X	O	X	O	X	X	O	X	O	X	O
		(RE,M,EC)		O		O			O		O		O			O		O		O
		(RE,EP,EC)																		
		(RE,M,EP,EC)																		
	Engine staging	Pipelined (k_d>1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_d=1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
	Workflow path	(RW,EC)	X	X	X	X	O	X	O	X	X	X	X	O	X	O	X	X	X	X
		(RW,M,EC)					O		O					O		O				
		(RW,EP,EC)																		
		(RW,M,EP,EC)																		
	Workflow staging	Pipelined (k_d>1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_d=1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Resource	Resource layout	Mediator	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
		Engine Repository	S	D	S	D	S	S	D	S	D	S	D	S	S	D	S	D	S	D
		Workflow Repository	S	S	E	E	S	S	S	S	S	E	E	S	S	S	S	S	E	E
		Engine execution	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
Integr.	Subject of integration	Auxiliary tool level																		
		Workflow engine level	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Interface	Backend	CLI	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		API																		
	Frontend	General	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Specific																		

Aspects			Proposed architectures																	
			PC19a, PC19b	PC20a, PC20b	PC21a, PC21b	PC22a, PC22b	PC23a, PC23b	PC24a, PC24b	PC25a, PC25b	PC26a, PC26b	PC27a, PC27b	PC28a, PC28b	PC29	PC30	PC31	PC32	PC33a, PC33b	PC34a, PC34b	PC35a, PC35b	PC36a, PC36b
Structure	Instance layout	(EP,M)																		
		(EP,RE)																		
		(EP,RW)						X	X					X				X		
		(EP,EC)																		
		(M,RE)			X		X		X			X								
		(M,RW)	X		X					X		X			X				X	
		(M,EC)																		
		(RW,RE)		X	X						X	X				X				X
		(RW,EC)														X				X
		(RE,EC)											X	X	X	X	X	X	X	X
	Type layout	δ_M	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		δ_{RE}	≥1	≥1	1	≥1	1	≥1	1	≥1	≥1	1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1
		δ_{RW}	1	≥1	1	≥1	≥1	≥1	≥1	1	≥1	1	≥1	≥1	1	≥1	≥1	≥1	1	≥1
		δ_{EC}	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1	≥1
Data flow	Bulk data path	(JP,JC)																		
		(JP,EC,JC)	X	X	X															
		(JP,EP,JC)																		
		(JP,M,JC)																		
		(JP,EP,EC,JC)	O*	O*	O*	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		(JP,M,EC,JC)																		
		(JP,EP,M,JC)																		
		(JP,EP,M,EC,JC)																		
	Bulk data staging	Pipelined (k_b=1)	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	X^a	O	O	O	O	X^a	X^a	X^a	X^a
		Not pipelined (k_b>1)	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	X^b	O	O	O	O	X^b	X^b	X^b	X^b
	Engine path	(RE,EC)	X	X	O	X	O	X	O	X	X	O	X	X	X	X	X	X	X	X
		(RE,M,EC)			O		O		O			O								
		(RE,EP,EC)																		
		(RE,M,EP,EC)																		
	Engine staging	Pipelined (k_d>1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_d=1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
	Workflow path	(RW,EC)	O	X	O	X	X	X	X	O	X	O	X	O	O	X	X	O	O	X
		(RW,M,EC)	O		O					O		O			O				O	
		(RW,EP,EC)												O				O		
		(RW,M,EP,EC)																		
	Workflow staging	Pipelined (k_d>1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
		Not pipelined (k_d=1)	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
Resource	Resource layout	Mediator	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
		Engine Repository	S	S	D	S	D	S	D	S	S	D	D	D	D	D	D	D	D	D
		Workflow Repository	S	S	S	S	S	E	E	S	S	S	S	E	S	S	S	E	S	S
		Engine execution	C	C	C	C	C	C	C	C	C	C	D	D	D	D	D	D	D	D
Integr.	Subject of integration	Auxiliary tool level																		
		Workflow engine level	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Interface	Backend	CLI	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		API																		
	Frontend	General	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
		Specific																		

Table 5.17: Existing and proposed WESW architectures, where E/D represents that the workflow repository nodes can be executed on dedicated (if $(EP, RW) \in h$) as well as on external machines (if $(EP, RW) \notin h$) and all other signs have the same purpose as in the case of table 5.13.

Workflow engines can be connected to the Gria service via their API, while WFBus also supports workflow engines with CLI. The subject of integration in both solutions is the workflow engine of the parent workflow system. On the one hand, both solutions utilize child engines that are deployed and executed on a dedicated machine. This is represented by instance layouts which have RE and EC coupled. On the other hand, none of the existing solutions provide workflow repositories. Hence, the workflow repository either can be hosted by the parent workflow engine or an external, decoupled workflow repository can be used. Based on these, two instance

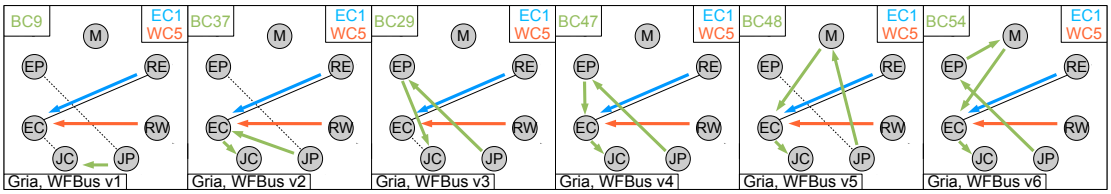

Figure 5.5: Combinations of possible Gria and WFBus based structures and data flow cases, where color coded arrows and lines represent the same concepts as in the case of figure 5.4.

layouts can be realised by the existing solutions: $h = \overline{\{(EP, RW), (RE, EC)\}}$ and $h = \overline{(RE, EC)}$. In terms of type layout, in the case of both solutions the mediator is hosted on a single machine, the child engines and workflow repositories can be hosted by multiple machines. Note that, since RE and EC are always coupled $\delta_{EC} = \delta_{RE}$. Since child engines are previously installed, they do not have to be transferred between machines. Hence, engine transfer path type is (RE, EC) and data staging is irrelevant. (See lemma A.3.) Workflows can be either transferred from the workflow repository directly, which is represented by engine path type (RW, EC), or they can be transferred via the mediator, which is represented by path type (RW, M, EC). The latter cannot be applied if an external workflow repository is used, since this is not supported by the mediators of the existing solutions. Bulk data is transferred via the mediator by default, but it is possible to realise further bulk data path types by passing references via the mediator if this is supported by the engines and/or the jobs. Bulk data path type (JP, JC) can be applied if the parent and the child workflow jobs can directly communicate (e.g via GridFTP). In this case bulk data staging is irrelevant. (JP, EC, JC) can be applied if the child workflow engine can directly gather the bulk data from the parent workflow job. From performance point of view, this path type is equivalent with (JP, JC) if the child job is executed locally to the child engine. (JP, EP, JC) can be applied if the child workflow job can directly gather the bulk data from the parent workflow

Gria, WFBus Architecture	Overhead	Overhead scalability	Latency	Latency scalability
v1	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}$	$O(rl_w)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}$	$O(rl_w)$
v2a	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+\frac{r(s_b+l_b)K}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+\frac{rs_bK}{\min\{r,\delta_{EC}\}}$	$O(rl_w)$
v2b	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2\frac{rl_bK}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+\frac{rl_bK}{\min\{r,\delta_{EC}\}}$	$O(rl_w+rl_b)$
v3a	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+s_bK$	$O(rl_w)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+s_bK$	$O(rl_w)$
v3b	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+l_bK$	$O(rl_w+l_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+l_bK$	$O(rl_w+l_b)$
v4a	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+s_bK+$ $+\frac{r(s_b+l_b)K}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+s_bK+\frac{rs_bK}{\min\{r,\delta_{EC}\}}$	$O(rl_w)$
v4b	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2\frac{rl_bK}{\min\{r,\delta_{JC}\}}$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+l_bK+\frac{rl_bK}{\min\{r,\delta_{EC}\}}$	$O(rl_w+rl_b)$
v5	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2rl_bK+$ $+\frac{rl_bK}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2rl_bK$	$O(rl_w+rl_b)$
v6	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2rl_bK+$ $+\frac{rl_bK}{\min\{r,\delta_{JC}\}}$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+l_bK+2rl_bK$	$O(rl_w+rl_b)$

Table 5.18: Performance characteristics of the Gria and WFBus-VRE Service based architectures, in the case when $(EC, JC), (EP, JP) \notin h$.

engine. This path type is equivalent with (JP, JC) if the parent job is executed locally to the parent engine. (JP, EP, EC, JC) can be applied if the workflow engines can directly communicate. This path type is equivalent with (JP, JC) if both jobs are executed locally to their workflow engines, equivalent with (JP, EC, JC) if the child job is executed locally, and equivalent with (JP, EP, JC) if the parent job is executed locally. (JP, M, EC, JC) can be applied if the parent job can directly communicate with the mediator. This path type is equivalent with (JP, M, JC) if the child job is executed locally. (JP, EP, M, EC, JC) can always be applied. This path type is equivalent with (JP, M, EC, JC) if the parent job is executed locally. Bulk data path type (JP, M, JC) and (JP, EP, M, JC) cannot be applied if the

Gria, WFBus Architecture	Overhead	Overhead scalability	Latency	Latency scalability
v1, v3a, v3b	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}$	$O\left(rl_w\right)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}$	$O\left(rl_w\right)$
v2a, v4a	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+\frac{r(s_b+l_b)K}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+\frac{rs_bK}{\min\{r,\delta_{EC}\}}$	$O\left(rl_w\right)$
v2b, v4b	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2\frac{rl_bK}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+\frac{rl_bK}{\min\{r,\delta_{EC}\}}$	$O(rl_w+rl_b)$
v5, v6	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2rl_bK+$ $+\frac{rl_bK}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2rl_bK$	$O(rl_w+rl_b)$

Table 5.19: Performance characteristics of the Gria and WFBus-VRE Service based architectures, in the case when $(EC, JC) \notin h \wedge (EP, JP) \in h$.

Gria, WFBus Architecture	Overhead	Overhead scalability	Latency	Latency scalability
v1, v2a, v2b	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+\frac{rl_bK}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}$	$O\left(rl_w\right)$
v3a, v4a	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+s_bK+$ $+\frac{rl_bK}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+s_bK$	$O\left(rl_w\right)$
v3b, v4b	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+\frac{rl_bK}{\min\{r,\delta_{JC}\}}$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+l_bK$	$O(rl_w+l_b)$
v5	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2rl_bK-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+rl_bK$	$O(rl_w+rl_b)$
v6	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2rl_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+rl_bK+l_bK$	$O(rl_w+rl_b)$

Table 5.20: Performance characteristics of the Gria and WFBus-VRE Service based architectures, in the case when $(EC, JC) \in h \wedge (EP, JP) \notin h$.

child job is executed remotely, because these are not supported by the mediators of the existing solutions. In the case of path types (JP, EC, JC), (JP, EP, JC), and (JP, EP, EC, JC) pipelined bulk data staging can be applied if this is supported by the engines involved in the transfer. In the case of (JP, M, EC, JC) and (JP, EP, M, EC, JC) data staging is not pipelined, since it is not supported by the mediators.

Gria, WFBus Architecture	Overhead	Overhead scalability	Latency	Latency scalability
v1, v2a, v3a, v4a, v2b,v3b, v4b	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+\frac{rl_bK}{\min\{r,\delta_{JC}\}}-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}$	$O(rl_w)$
v5, v6	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+2rl_bK-l_bK$	$O(rl_w+rl_b)$	$\frac{rl_wK}{\min\{r,\delta_{RW},\delta_{EC}\}}+$ $+rl_bK$	$O(rl_w+rl_b)$

Table 5.21: Performance characteristics of the Gria and WFBus-VRE Service based architectures, in the case when $(EC, JC), (EP, JP) \in h$.

All architectures that can be realised by the existing solutions are included in table 5.17. Combinations of the different structures and representative data flow cases are illustrated in figure 5.5 and performance characteristics of theses architectures are described in table 5.18-5.21.

5.4.2 Proposed WESW solutions

WESW architecture analysis (see section 5.3) identified several architectures that can be proposed in different cases. Recommended non-performance related architectural aspects (interface and integration) are identified in section 5.3.1. According to this section recommended frontend and backend interfaces are the same as in the case of WESAs: generic and CLI respectively. Recommended subject of integration is the workflow engine of the parent workflow since this enables both dynamic and semi-dynamic invocation. Numerous proposed structure and data flow combinations are identified in section 5.3.5. These are specified in table 5.13. Based on the properties of the parent and child workflow and the child workflow engine 54 different cases were identified and described in this section. Table 5.14 summarises which structure and data flow combinations are recommended in each case. Based on these all proposed architectures are defined in table 5.17 along with the architectures that can be realised by the Gria service and WFBus based solutions. The recommended WESW solution should realise one or more of the proposed architectures depending

on what cases it will be used in.

5.4.3 Comparison of existing and proposed WESW solutions

Just like in the case of WESAs, the main difference between the proposed and existing architectures lies in their structure. While the Gria and WFBus based architectures invoke previously installed workflow engines in a service oriented manner, the proposed solutions also support submitting the workflow engines to computational Grid resources.

Overhead on bulk data transfer is the most important indicator of performance. This performance property is strongly affected by where the parent and child workflow jobs are executed and whether they transfer data directly. These properties are described and detailed in paragraph *1. Parent workflow jobs* and *2. Child workflow jobs* in section 5.3.5. According to these:

- case 1/a and 2/a represent (respectively) that the parent and child jobs are executed locally to their engines;
- case 1/b-i and 2/b-i represent (respectively) that the parent and child jobs are executed remotely to their engines and data can be transferred to/from these jobs without transferring data via their engines; and
- case 1/b-ii and 2/b-ii represent (respectively) that the parent and child jobs are executed remotely to their engines and data can only be transferred to/from these jobs via their engines.

Based on these, nine different cases were identified. The comparison of overhead scalability of the proposed end existing architectures can be seen in table 5.22. The

Case	Proposed architecture	Ovh. scl. Pip	Ovh. scl $\neg Pip$	Gria, WFBus architecture	Ovh. scl. Pip	Ovh. scl $\neg Pip$
1/a, 2/a	PC1-PC7	$O(0)$	$O(0)$	v1-v6	$O(rl_b)$	$O(rl_b)$
1/a, 2/b-i	PC29-P32	$O(0)$	$O(0)$	v1,v3	$O(0)$	$O(0)$
1/a, 2/b-ii	PC15-P21	$O(1)$	$O(l_b)$	v2,v4-v6	$O(rl_b)$	$O(rl_b)$
1/b-i, 2/a	PC1-PC7	$O(0)$	$O(0)$	v1, v2, v5	$O(rl_b)$	$O(rl_b)$
1/b-i, 2/b-i	PC29-P32	$O(0)$	$O(0)$	v1	$O(0)$	$O(0)$
1/b-i, 2/b-ii	PC15-P21	$O(1)$	$O(l_b)$	v2, v5	$O(rl_b)$	$O(rl_b)$
1/b-ii, 2/a	PC8-PC14	$O(1)$	$O(l_b)$	v3, v4, v6	$O(rl_b)$	$O(rl_b)$
1/b-ii, 2/b-i	PC33-P36	$O(1)$	$O(l_b)$	v3	$O(1)$	$O(l_b)$
1/b-ii, 2/b-ii	PC22-P28	$O(1)$	$O(l_b)$	v4, v6	$O(rl_b)$	$O(rl_b)$

Table 5.22: Bulk data transfer overhead scalability of the proposed and existing WESW architectures in different cases, where Pip represents pipelined and $\neg Pip$ represents non pipelined bulk data staging.

table also describes which architectures are proposed in the different cases and also which architectures of the existing solutions are recommended in those cases.

In case the case of 2/b-i proposed architectures invoke previously installed workflow engines. This means that overhead and latency are not increased by the transfer time of the workflow engine. In this case the overall overhead and latency of the existing and proposed solutions are identical.

In all other cases, proposed architectures submit workflow engines to computational resources for execution. This increases overhead with engine transfer time, but provides lower overhead on bulk data transfer. In the case of the proposed solutions, overhead on bulk data staging is either 0 or constant ($O(1)$) if pipelined bulk data transfer is possible. It is linear with bulk data amount, but independent of the number of simultaneous requests ($O(l_b)$) if pipelined bulk data staging is not possible. In the case of the existing solutions overhead on bulk data transfer is linear with both bulk data amount and the number of simultaneous requests ($O(rl_b)$). Therefore, proposed solutions provide significantly lower overall overhead and latency than the existing solutions especially in the case of large numbers of

simultaneous requests. For detailed performance figures see table 5.15, 5.16 5.18, 5.19, 5.20 and 5.21.

The performance improvements of proposed architectures in the case of WESWs are similar to the WESA performance improvements described in section 4.4.3, where graphical representation is also provided.

In addition, in the case of the Gria based architectures engines are accessed via APIs, which means that programming knowledge is required to add a new workflow engine to the system. Engines can be connected to WFBus both via CLI or API. In the case of the proposed concepts, engines are connected only via CLI. This means that user level knowledge is sufficient to add a new engine to the system and also means that the engine can be submitted to computational resources.

5.5 Implementation

Several proposed WESW architectures were implemented based on GEMLCA and P-GRADE. The implementation is partially based on the concept of DASW and WESA implementation described in section 3.5 and 3.5, and enables P-GRADE workflows (as parent workflows) to embed and execute Kepler, Moteur, Taverna, and Triana workflows (as child workflows). Note that P-GRADE to be the parent workflow system was chosen, because it can directly interface with GEMLCA. GIB extension (see description in section 4.5) of GEMLCA is used to enable the deployment and execution of workflows via GEMLCA.

The GEMLCA client is integrated with the workflow engine and legacy applications executed by GEMLCA are represented as P-GRADE jobs. Figure 5.6 illustrates how a Triana workflow represented as a job can be parametrised in the P-GRADE workflow editor. First, the Grid and the GEMLCA service that hosts

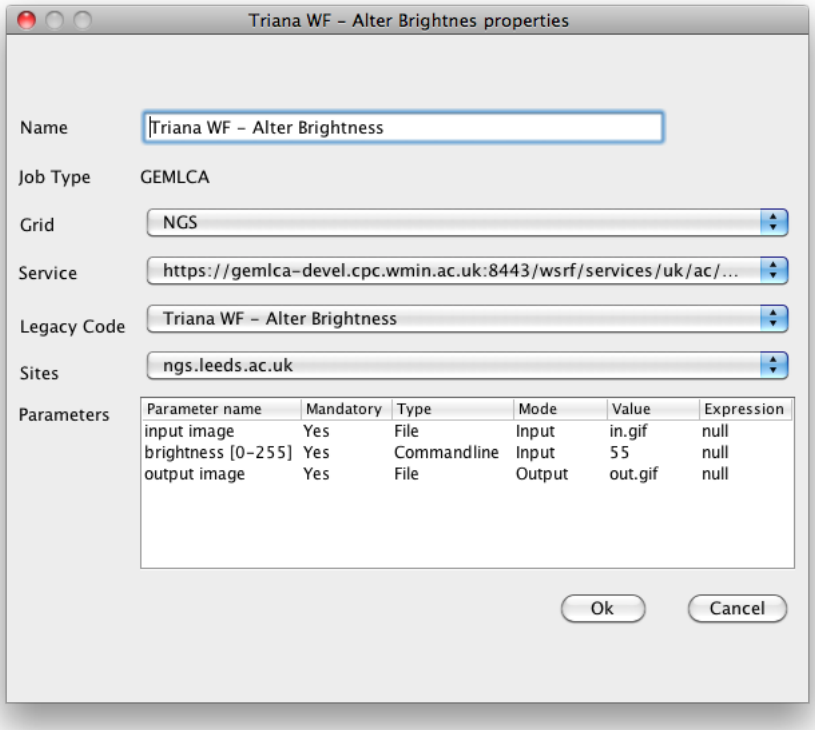

Figure 5.6: Parametrisation of a Triana child workflow in a P-GRADE parent workflow

the desired workflow have to be specified. Next, the workflow (legacy code) has to be selected and the computational resource (site) where it will be executed. After this, a parameter table pops up, where the user can specify the workflow arguments.

Figure 5.7 illustrates how PC26 is implemented by GEMLCA based WESW solution. Similarly to the GEMLCA based WESA implementation, in this solution the GEMLCA service realises the mediator and also the workflow repository and a GridFTP storage machine realises the engine repository. This means that the mediator is coupled with the workflow repository and the engine repository is decoupled. When the job which represents the child workflow is to be executed, the P-GRADE engine gathers the bulk data from the previous job in the P-GRADE workflow.

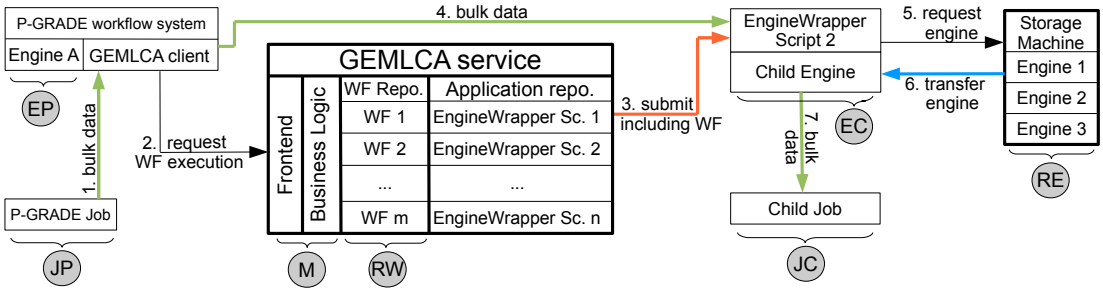

Figure 5.7: Implementation of WESW PC26 based on GEMLCA, where black arrows represent control data, blue arrow represents engine transfer, yellow arrow represents workflow transfer, and green arrows represent bulk data transfer.

(This is not illustrated in the figure.) Next the P-GRADE engine passes a request to the local GEMLCA client. This request includes all information specified in the parameter table shown in figure 5.6 and the following steps are the same as in the case of the GEMLCA based WESA implementation described in section 4.5. The solution can also implement PC24 if the child workflow is provided by the P-GRADE parent workflow engine.

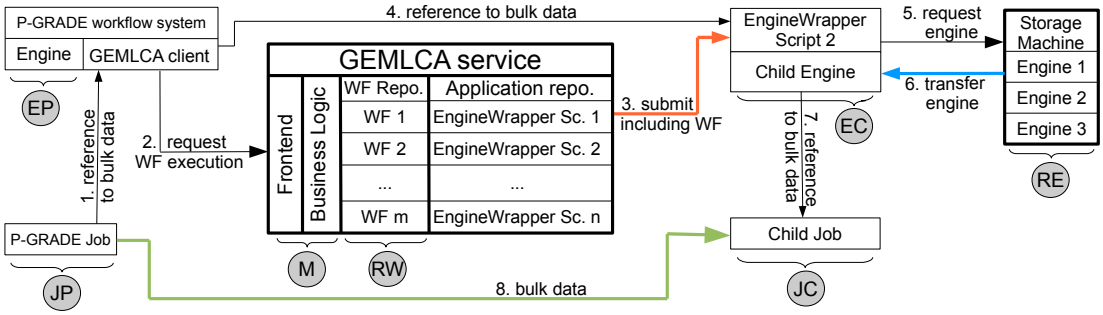

Figure 5.8: Implementation of WESW PC5 based on GEMLCA

If it is possible for the child job to gather data directly from the machine of the parent job (i.e. using GridFTP), only a reference to the bulk data should be passed via GEMLCA and bulk data should be transferred directly between the parent and child jobs. See illustration on figure 5.8. If this is the case, then PC5 (child workflow

is in GEMLCA application repository) and PC3 (child workflow is provided by the parent workflow) can be implemented as well. Note that this approach works only if the parent workflow is executed on such Grid (i.e. the UK NGS), where data is not erased automatically after job execution, therefore, it can be gathered later on by the child workflow. Similarly, if the child engine can gather the bulk directly data from the parent job using a reference, then PC19 can also be implemented.

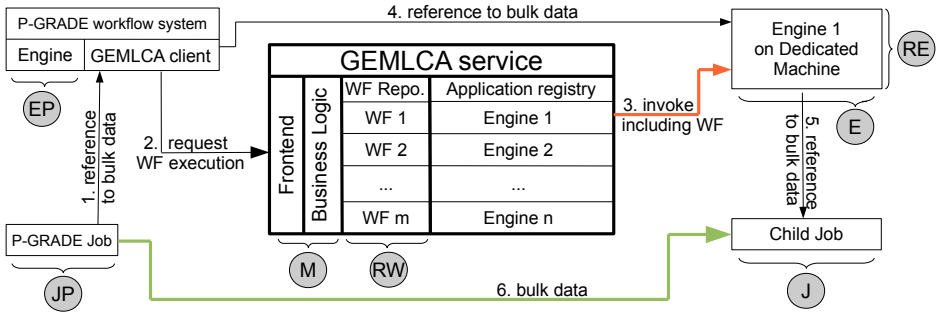

Figure 5.9: Implementation of WESW PC31 based on GEMLCA

As it was described in section 4.5 the four workflow engines were also deployed on dedicated machines. If the child job can gather bulk data from the machine of the parent job then PC31 (if workflow is in GEMLCA application repository) and PC30 (workflow is provided by the parent workflow) can also be implemented. See illustration in figure 5.9.

The solution was tested on both Globus and gLite with the same cases described in section 4.5.

5.6 Summary

This chapter proposed 36 WESW architectures in order to realise workflow interoperability at the level of engine integration. Architectures were selected based on the mathematical model introduced in chapter 2. The analysis primarily focuses on the

performance characteristics of the different architectures, but other properties such as generality and extendibility were also considered.

Two existing solutions the Gria service based approach and VLE-WFBus realise workflow interoperability at the level of workflow engines. They are both based on the same concept, where workflow engines are previously deployed on dedicated machines and accessed via a centralised service, which may bottleneck bulk data transfer in the case of simultaneous requests. This can be avoided only in special cases.

Similarly to the proposed WESA architectures, the novelty of the proposed WESW architectures is that engines are distributed across the available computational machines provided by the Grid. Performance characteristics of a WESW architecture are strongly affected by where the parent and child workflow jobs are executed and whether they can exchange data directly. The analysis recommends different architectures under different circumstances considering these and further aspects.

The P-GRADE and GEMLCA based reference implementation realises 7 of the proposed architectures. The GEMLCA based WESW solution has the same limitations as the GEMLCA based WESA solution with the addition that direct data transfer between parent and child workflow jobs can only be achieved if they support the same protocol (i.e. GridFTP) for transferring data. This solution is general and easily extendible for the same reasons as the similar GEMCLA based WESA reference implementations.

Conclusions

The contribution this thesis is aiming to make to scientific knowledge lies in its proposed architectures in two areas: executing heterogeneous workflows and accessing heterogeneous data resources.

On the one hand, existing workflow systems are based on different technologies. Therefore, to achieve interoperability between their workflows at any level is a challenging task. In spite of the fact that there is a clear demand for interoperable workflows, since it enables scientists to share workflows, build on top of the existing work of others, and to create multi-disciplinary workflows, there are only limited, ad-hoc workflow interoperability solutions. These solutions realise workflow interoperability between a small set of workflow systems and do not consider performance issues that arise in the case of large-scale scientific workflows. Scientific workflows are typically computation and/or data intensive and are executed in a distributed environment to speed up their execution time. Therefore, their performance is a key issue. Existing interoperability solutions bottleneck the communication between workflows in most scenarios dramatically increasing execution time.

On the other hand, even though most scientific computational experiments are based on data that reside in databases, few have a very limited support to access databases and other types of data resources. Therefore, there is a demand for a solution that provides access to a large set of data resources. If such a solution is

general, in the sense that it can be adopted by several workflow systems, then it also enables workflows of different systems to access the same data resources and therefore interoperate at data level. For the same reasons as described above, the performance characteristics of such a solution are inevitably important. Although in terms of functionality, there are solutions which could be adopted by workflow systems for this purpose, they provide poor performance. This is the reason why they did not gain wide acceptance by the scientific workflow community.

The main objective of this thesis is to propose architectures for two major problems of currently existing workflow systems: *workflow interoperability* at the level of workflow engines and *data access*. It proposes a set of architectures to realise heterogeneous data access solutions and to realise heterogeneous workflow execution solutions. The primary goal was to investigate how such solutions can be implemented and integrated with workflow systems. The secondary goal was to analyse how such solutions can be implemented and utilised by single applications. Based on these the following four areas were identified:

- **DASG** heterogeneous Data Access Solutions for Grid applications
- **DASW** heterogeneous Data Access Solutions for Workflows
- **WESA** heterogeneous Workflow Execution Solutions for Applications
- **WESW** heterogeneous Workflow Execution Solutions for Workflows - workflow nesting

In the case of both data access solutions and workflow execution solutions it was important to distinguish between whether the access/execution solution is provided for applications or workflows. The reason for that is that workflows are far more complex then applications executed on a single machine. In many cases workflows

are distributed and their execution is controlled by a remote workflow engine. This can dramatically increase the complexity of the problem.

The mathematical model designed for analysing the different architectures and their properties, existing and proposed solutions is described in chapter 2. In order to clearly show the difference in performance of the compared architectures, they have to be analysed based on the same network environment. Although, in real life machines and the network that connects them have different performance characteristics, the mathematical model used for analysing the different architectures assumes an idealised homogeneous network where all machines are connected via a full-duplex network with constant bandwidth. Axioms of this homogeneous network can be found in section 2.3.2. In a real life scenario available bandwidth changes dynamically with data traffic generated by third party machines and software. Therefore, modelling such a heterogeneous network dramatically increases the complexity of the analysis and involves several unknown variables. Analysis based on a heterogeneous network environment or real life measurements can be covered by future work, but this is out of the scope of this thesis.

The model provides concepts for defining different architectures including the distribution of different software components between the available machines and data transfer between these software components. It also provides a set of functions to analyse the performance characteristics of the architectures.

Performance of the different architectures are compared based on a set of analytical scenarios. A scenario not only defines the performance related properties of an architecture, but it also defines the number and size of the different data-sets that need to be exchanged between the different software components. In each case the set of analytical scenarios were chosen based on the following principles. An analytical scenario: has to be suitable to indicate the performance differences of the different architectures; has to be relevant in the sense that it has to cover typical

Area	Proposed architecture sets	Implemented architecture sets
DASG	2	1
DASW	14	1
WESA	18	6
WESW	36	7

Table 5.23: Number of proposed and implemented architecture sets in different areas

user scenarios; and has to be simple enough for evaluation within the scope of this thesis. The selected analytical scenarios on which the analysis is based fulfil the above requirements. Note that it is not aimed by this thesis to analyse the different architectures based on specific scenarios. This can be covered by future work using a similar approach presented in the case of selected analytical scenarios.

This thesis proposes 70 disjunctive sets of architectures: 2 in the case of DASGs (heterogeneous Data Access Solutions provided for Grid applications); 14 in the case of DASWs (heterogeneous Data Access Solutions provided for Workflows); 18 in the case of WESAs (heterogeneous Workflow Execution Solutions provided for Applications); and 36 in the case of WESWs (heterogeneous Workflow Execution Solutions provided for Workflows - workflow nesting) See illustration in table 5.23.

The key novelty in the case of most proposed architectures is that computational and storage machines provided by the Grid are utilised in order to divide the load of the different software components. This can be achieved by dynamically distributing these software components between the available machines. This way highly scalable architectures can be realised that deliver data between these components with low overhead. Dynamically distributing software components on the Grid requires that the software dependencies of the given component are fulfilled on the computational machine where it is executed. If this is not provided, software dependencies have to be statically linked.

Backend software components are data resource clients (i.e. MySQL client, SRB client, GridFTP client, etc.) in the case of DASGs and DASWs and workflow engines (i.e. MOTEUR engine, Taverna engine, Triana Engine) in the case of WESAs and WESWs. All proposed architectures are easily extendible thanks to the proposed backend interfaces which enable the extension of the available backend components via CLI (Command Line Interface). Since this approach does not connect the backend software components via API (Application Programming Interface) no programming skills are needed, user level understanding of the given backend is sufficient. The limitation of this approach is that it can be used only for data resource clients and workflow engines that provide a CLI. However, in most cases this is provided.

All proposed architectures are generic since the proposed frontends do not restrict the type and number of parameters that can be passed to the desired backend. Since these parameters have to be fed to the backend component as command line parameters, they have to be passed either as files or as command line arguments. Note that this limitation does not mean that all data has to be passed this way to the backend component. For instance, if a workflow engine can gather information from a web service, the endpoint reference can be passed to the engine via its CLI, and the engine can gather any type of data from the given web service.

Since architectures are proposed based on a theoretical methodology, it was also important to demonstrate that the proposed concepts are valid and possible to implement. 15 of the proposed architecture sets were implemented based on the GEMLCA application repository and submitter: 1 in the case of DASGs; 1 in the case of DASWs; 6 in the case of WESAs; and 7 in the case of WESWs. These numbers are also illustrated in table 5.23. Application deployment in GEMLCA is straightforward, it can be performed using the graphical user interface of the GEMLCA administrator portlet. This only requires the definition of the CLI of the

given application (data resource client in the case of DASGs and DASWs or workflow engine in the case of WESAs and WESWs) and specification how and where it can be executed. In the case of DASGs and DASWs, a MySQL client, while in the case of WESAs and WESWs, Taverna, Triana, Kepler, and MOTEUR workflow engines were deployed in GEMLCA. Currently GEMLCA supports execution on gLite, GT2 and GT4 based Grids. However, thanks to its modular architecture this can be simply extended by adding further submitter plugins to GEMLCA. Since GEMCLA is based on a command line approach, execution monitoring is only possible via the standard output and error messages of the running applications.

Partially based on the presented work related to workflow execution and interoperability (WESAs and WESWs), the European Union FP7 funded SHIWA project started on the 1st of July 2010 and lasts two years. Its main goal is to leverage existing workflow based solutions and enable workflow interoperability at different levels. Fine-grained approach is aiming to realise language level interoperability by defining an intermediate workflow representation that can be used for translation of workflows across different workflow systems. Coarse-grained approach is aiming to realise engine level interoperability, partially based on the concepts and the GEMCLA based reference implementations presented in this thesis.

Within the scope of the SHIWA project, the GEMLCA based workflow execution solutions will be extended with the support of further workflow systems both at parent and child workflow side. It will also be extended with a functionality rich workflow repository that will enable scientists to upload and share their workflows, browse and download workflows of others. These solutions will be used and tested by the SHIWA user community in real user scenarios. Results and limitations will be published in forthcoming papers. Although the problem of accessing heterogeneous data resources is not in the main focus of SHIWA, it is an important issue that has to be addressed. The proposed data access solutions are available for the users of

the NGS via the NGS-PGRADE portal, their concept will be published in the near future.

Contributions of this thesis are aiming to ease the work of scientists and the workflow community. The presented mathematical model makes it possible to analyse a wast range of possibilities and identify optimal architectures. Reference implementations show that the proposed concepts are valid and possible to realise. We believe that the presented work will help scientists to exploit the potential of workflows and Grids and will enable the collaboration of the scientific community to address grand challenges that were not solved before.

Dissemination of the research findings

An essential component of the research is the dissemination of its findings. Five papers have been published so far in this topic and further journal articles will be published summarizing the findings of this research. Furthermore, partially based on the presented work related to workflow execution and interoperability (WESAs and WESWs), the European Union FP7 funded SHIWA project started on the 1st of July 2010 and lasts two years. Its main goal is to leverage existing workflow based solutions and enable workflow interoperability at different levels.

List of publications

1. Kiss, T.; Kukla, T. and Terstyanszky, G.: Towards Grid data interoperation: OGSA-DAI data resources in computational Grid workflows, *CoreGRID Integration Workshop proceedings*, 2008

2. Kiss, T. and Kukla, T.: High-level User Interface for Accessing Database Resources on the Grid, *7th International Conference on Distributed and Parallel Systems*, 2008

3. Kukla, T.; Kiss, T.; Terstyanszky, G. and Kacsuk, P.: A General and Scalable

Solution for Heterogeneous Workflow Invocation and Nesting, *Supercomputing proceedings*, 2008

4. Kukla, T.; Kiss, T.; Terstyanszky, G. and Kacsuk, P.: Integrating OGSA-DAI with computational Grid workflows, *Philosophical Transactions A of the Royal Society*, 2009

5. Kiss, T.; Kukla, T.; Terstyanszky, G. and Kacsuk, P.: Achieving Interoperation of Grid Data Resources via Workflow Level Integration, *Journal of Grid Computing*, 2009

Appendix A

Proofs

Definition A.1 (Structure generated by an instance layout)
Let $\gamma_I : \mathcal{L}_I(\mathcal{T}) \to \mathscr{G}_r$ be a function that maps a structure to any instance layout, where the structure contains all couplings defined by a particular instance layout as:

$$\gamma_I(h) := \bigcup_{i=1}^{r} \{(\varphi_i(t_1), \varphi_i(t_2)) \in \mathcal{N}_i^2 \| (t_1, t_2) \in h)\} \tag{A.1}$$

Note that since, $\forall i \in [1..r] : \varphi_i$ is a bijection between $\mathcal{T}$ and $\mathcal{N}_i$ and h is an equivalence relation over $\mathcal{T}$, $\{(\varphi_i(t_1), \varphi_i(t_2)) \in \mathcal{N}_i^2 \| (t_1, t_2) \in h)\}$ is also an equivalence relation over $\mathcal{N}_i$. Furthermore, since these equivalence relations are defined over distinct sets, their union ($\gamma_I(h)$) is also an equivalence relation over $\mathcal{N}$, and as such it is a structure over $\mathcal{N}$, also meaning that $\gamma_I(h) \in \mathscr{G}_r$. Based on definition 2.11, $\forall r \in \mathbb{N}^+, h \in \mathcal{L}_I(\mathcal{T}) : \gamma_I(h)$ implements h.

Definition A.2 (Structure generated by a type layout)
Let $\gamma_T : \mathcal{L}_T(\mathcal{T}) \to \mathscr{G}^*$ be a function that maps a structure to any type layout. The structure contains all couplings defined by a particular type layout as:

$$\gamma_T(\delta) := \bigcup_{t \in \mathcal{T}} \{(\varphi_i(t), \varphi_j(t)) \in \mathcal{N}_t^2 \| i, j \in [1..r] \wedge i \equiv j \mod \delta_t\}. \tag{A.2}$$

Note that $\forall t \in \mathcal{T} : \{(\varphi_i(t), \varphi_j(t)) \in \mathcal{N}_t^2 \| i, j \in [1..r] \wedge i \equiv j \mod \delta_t\}$ is an equivalence relation over $\mathcal{N}_t$. Moreover, since these are distinct sets, $\gamma_T(h)$ is also an equivalence relation, and as such it is a structure over $\mathcal{N}$, also meaning that $\gamma_T(h) \in \mathscr{G}_r$. Based on definition 2.13, $\forall r \in \mathbb{N}^+, \delta \in \mathcal{L}_T(\mathcal{T}) : \gamma_T(\delta)$ implements δ.

Lemma A.1 (Structure layout implementation)
Having $h \in \mathcal{L}_I(\mathcal{T}), \delta \in \mathcal{L}_T(\mathcal{T})$, the following statement is true:

$$\forall (t_1, t_2) \in h : \delta_{t_1} = \delta_{t_2} \Rightarrow \exists G \in \mathscr{G}_r \text{ that implements structure layout } (h, \delta) \quad \text{(A.3)}$$

Proof G implements (h, δ) if and only if it implements both h and δ. (See definition 2.15.) According to definition 2.11 and 2.13 G implements both h and δ if and only if:

$$\left.\begin{aligned} \forall i \in [1..r], (t_1, t_2) \in \mathcal{T}^2 : \ (t_1, t_2) \in h \Leftrightarrow (\varphi_i(t_1), \varphi_i(t_2)) \in G, \text{ and} \quad \text{(a)} \\ \forall i, j \in [1..r], t \in \mathcal{T} : \ i \equiv j \mod \delta_t \Leftrightarrow (\varphi_i(t), \varphi_j(t)) \in G. \quad \text{(b)} \end{aligned}\right\} \quad \text{(A.4)}$$

Let $G_I := \gamma_I(h)$, $G_T := \gamma_T(\delta)$, and $G := \overline{G_I \bigcup G_T}$. Note that G_I implements h and G_T implements δ based on their definition. First, by applying mathematical induction, it is showed that:

$$\begin{aligned} &\forall i, j \in [1..r], t_1, t_2 \in \mathcal{T} : \\ &\quad (\varphi_i(t_1), \varphi_j(t_2)) \in G \Rightarrow \forall m \in [1..r] : (\varphi_m(t_1), \varphi_m(t_2)) \in G_I \end{aligned} \quad \text{(A.5)}$$

Since both G_I and G_T are symmetric and reflexive, $G_I \bigcup G_T$ is symmetric and reflexive as well. This implies that $\overline{G_I \bigcup G_T}$ is the transitive closure of $G_I \bigcup G_T$, because the transitive closure of a symmetric, reflexive relation is symmetric and reflexive. The transitive closure of G equals to its connectivity relation which can be generated as:

$$G = \bigcup_{u=0}^{|E_0|} E_u, \text{ where} \quad \text{(A.6)}$$

$$E_0 := G_I \bigcup G_T, \text{ and } \forall u \in \mathbb{N}^+ : E_u := E_{u-1} \bigcup (E_{u-1} \circ E_0). \quad \text{(A.7)}$$

In the case of E_0 the following is always true:

$$\forall i, j \in [1..r], t_1, t_2 \in \mathcal{T} : (\varphi_i(t_1), \varphi_j(t_2)) \in E_0 \Rightarrow i = j \vee t_1 = t_2 \tag{A.8}$$

$$(\varphi_i(t_1), \varphi_j(t_2)) \in E_0 \Rightarrow \forall m \in [1..r] : (\varphi_m(t_1), \varphi_m(t_2)) \in G_I \tag{A.9}$$

Statement A.8 is true, since $\forall i, j \in [1..r], t_1, t_2 \in \mathcal{T}' : (\varphi_i(t_1), \varphi_j(t_2)) \in G_I \Rightarrow i = j \wedge (\varphi_i(t_1), \varphi_j(t_2)) \in G_T \Rightarrow t_1 = t_2$ meaning that G_I does not define couplings between the nodes of different instances and G_T does not define couplings between the nodes of different node types. This implies statement A.9. Because, if $i = j$, then $(\varphi_i(t_1), \varphi_i(t_2)) \in G_I \Leftrightarrow \forall m \in [1..r] : (\varphi_m(t_1), \varphi_m(t_2)) \in G_I$ based on definition A.1. If $t_1 = t_2$, then $\forall m \in [1..r] : (\varphi_m(t_1), \varphi_m(t_2)) \in G_I$ is true, because G_I is reflexive. Assume that $\forall v \in [0..u]$:

$$\begin{aligned} &\forall i, j \in [1..r], t_1, t_2 \in \mathcal{T} : \\ &\quad (\varphi_i(t_1), \varphi_j(t_2)) \in E_v \Rightarrow \forall m \in [1..r] : (\varphi_m(t_1), \varphi_m(t_2)) \in G_I. \end{aligned} \tag{A.10}$$

Based on statement A.7:

$$\begin{aligned} \forall i, j \in [1..r], t_1, t_2 \in \mathcal{T} :&(\varphi_i(t_1), \varphi_j(t_2)) \in E_{u+1} \Rightarrow \\ &(\varphi_i(t_1), \varphi_j(t_2)) \in E_u \vee (\varphi_i(t_1), \varphi_j(t_2)) \in E_u \circ E_o \end{aligned} \tag{A.11}$$

If $(\varphi_i(t_1), \varphi_j(t_2)) \in E_u$ then based on assumption A.10 $\forall m \in [1..r] : (\varphi_m(t_1), \varphi_m(t_2)) \in G_I$. If $(\varphi_i(t_1), \varphi_j(t_2)) \in E_u \circ E_o$ then based on the definition of the composition of binary relations:

$$\exists k \in [1..r], t_3 \in \mathcal{T} : (\varphi_i(t_1), \varphi_k(t_3)) \in E_u \wedge (\varphi_k(t_3), \varphi_j(t_2)) \in E_0 \tag{A.12}$$

which implies that:

$$\forall m \in [1..r] : (\varphi_m(t_1), \varphi_m(t_3)), (\varphi_m(t_3), \varphi_m(t_2)) \in G_I \Rightarrow \tag{A.13}$$

$$\Rightarrow \forall m \in [1..r] : (\varphi_m(t_1), \varphi_m(t_2)) \in G_I, \tag{A.14}$$

because G_I is transitive. Having these, it can be stated that statement A.5 is always true, which implies that:

$$\forall i \in [1..r], t_1, t_2 \in \mathcal{T} : (\varphi_i(t_1), \varphi_i(t_2)) \in G \Leftrightarrow (\varphi_i(t_1), \varphi_i(t_2)) \in G_I. \tag{A.15}$$

This means that G always implements h. Next it is showed under what conditions G implements δ. Because of statement A.5 and the fact that G is transitive:

$$\begin{aligned}\forall i,j \in [1..r], t_1,t_2 \in \mathcal{T} :&(\varphi_i(t_1),\varphi_j(t_2)) \in G \Rightarrow \\ &\Rightarrow (\varphi_i(t_1),\varphi_j(t_1)),(\varphi_i(t_2),\varphi_j(t_2)) \in G_T\end{aligned} \tag{A.16}$$

Based on definition A.2, the followings must be true:

$$(\varphi_i(t_1),\varphi_j(t_1)) \in G_T \Leftrightarrow i \equiv j \mod \delta_{t_1} \tag{A.17}$$

$$(\varphi_i(t_2),\varphi_j(t_2)) \in G_T \Leftrightarrow i \equiv j \mod \delta_{t_2} \tag{A.18}$$

These are ensured in the following two cases:

$$\delta_{t_1},\delta_{t_2} \geqq r \Rightarrow (i \equiv j \mod \delta_{t_1} \Leftrightarrow i \equiv j \mod \delta_{t_2}) \tag{A.19}$$

$$\delta_{t_1} = \delta_{t_2} \Rightarrow (i \equiv j \mod \delta_{t_1} \Leftrightarrow i \equiv j \mod \delta_{t_2}) \tag{A.20}$$

But they are false in any other case:

$$\delta_{t_1} \neq \delta_{t_2} \wedge \delta_{t_2} < r \Rightarrow \exists i,j \in [1..r] : i \equiv j \mod \delta_{t_1} \wedge i \not\equiv j \mod \delta_{t_2} \tag{A.21}$$

$$\delta_{t_1} \neq \delta_{t_2} \wedge \delta_{t_1} < r \Rightarrow \exists i,j \in [1..r] : i \not\equiv j \mod \delta_{t_1} \wedge i \equiv j \mod \delta_{t_2} \tag{A.22}$$

This means that G implements δ if and only if $\forall t_1,t_2 \in \mathcal{T} : \delta_{t_1},\delta_{t_2} > r \vee \delta_{t_1} = \delta_{t_2}$. Based on these, we can conclude that statement A.4 is always true independently from r. ∎

Lemma A.2 (Time of transferring a byte array sequence via a path *)
By applying mathematical induction based on definition 2.26, it can be proven that time of transferring byte array sequence $(x_1, x_2, ..., x_k)$ via path $p = (n_0, n_1, ..., n_m)$ equals to:

$$\sum_{j=1}^{m} \tau_e(s,(n_{j-1},n_j)) + (k-1)\max_{j=1}^{m} \tau_e(s,(n_{j-1},n_j)). \tag{A.23}$$

Proof Definition 2.26 defines when a byte array can be transferred between two neighbouring nodes of the node sequence and definition 2.45 defines how much time it takes to transfer a byte array between any two nodes. Based on these, first, x_1 is transferred from n_0 to n_1, that takes $\tau_e(\lambda(x_1), (n_o, n_1))$ time. Than x_1 is transferred from n_1 to n_2, this takes $\tau_e(\lambda(x_1), (n_1, n_2))$ time. Next, it is transferred to n_3, n_4, and so on, until it reaches n_m. Thus, transferring x_1 to n_j takes: $\sum_{u=1}^{j} \tau_e(\lambda(x_1), (n_{u-1}, n_u))$ time ($j \in [1..m]$). As soon as x_1 is transferred to n_1, the transfer of x_2 from n_0 to n_1 starts. When it is finished, x_3 is transferred from n_0 to n_1, and so on. In general, it takes $\sum_{u=1}^{i} \tau_e(\lambda(x_u), (n_0, n_1))$ time, for x_i to reach n_1 ($i \in [1..k]$).

However, the question is how much time does it take to transfer the whole byte array sequence from n_0 to n_m, which is the same as the time required from start for x_k to reach n_m. To calculate this value, let $A = [a_{i,j}]_{m \times k}$ ($a_{i,j} \in \mathbb{R}_0^+$) be a matrix, where $a_{i,j}$ shows how much time is needed from start for x_i to reach n_j. Based on the above, the first row and the first column of this matrix can be constructed respectively as:

$$\forall j \in [1..k] : a_{1,j} = \sum_{u=1}^{j} \tau_e(\lambda(x_1), (n_{u-1}, n_u)) \text{ and} \tag{A.24}$$

$$\forall i \in [1..m] : a_{i,1} = \sum_{u=1}^{i} \tau_e(\lambda(x_u), (n_0, n_1)). \tag{A.25}$$

According to definition 2.26, in the case when $i \in [2..k]$ and $j \in [2..m]$, transfer of x_i between n_{j-1} and n_j is performed only when the transfer of x_i between n_{j-2} and n_{j-1} and the transfer of x_{i-1} between n_{j-1} and n_j are both finished. Therefore, the rest of the matrix can be constructed as:

$$\forall i \in [2..n], j \in [2..m] : a_{i,j} = \max\{a_{i-1,j}, a_{i,j-1}\} + \tau_e(\lambda(x_i), (n_{j-1}, n_j)). \tag{A.26}$$

Since, it is assumed that $\forall i \in [1..k] : \lambda(x_i) = s$, therefore, $\forall i \in [1..k], j \in [1..m] : \tau_e(\lambda(x_i), (n_{j-1}, n_j)) = \tau_e(s, (n_{j-1}, n_j))$. By applying mathematical induction, it can

be proven, that

$$a_{n,m} = \sum_{j=1}^{m} \tau_e(s, (n_{j-1}, n_j)) + (k-1) \max_{j=1}^{m} \tau_e(s, (n_{j-1}, n_j)). \tag{A.27}$$

This amount of time is required for the last byte array, to reach node n_m. Therefore, this amount of time is required to transfer the whole byte array sequence via path p. ∎

Lemma A.3 (Slice size independence)

If $x \in \mathcal{B}$, $p \in \mathcal{P}$, and path p has only two elements, then $\forall s_1, s_2 \in \mathbb{N}^+$, where $\lambda(x)$ is dividable by s_1, s_2:

$$\tau_p(\lambda(x), s_1, p) = \tau_p(\lambda(x), s_2, p). \tag{A.28}$$

Proof This is implied by definition 2.46, since if $p = (n, m)$ and G is the structure of the nodes, then

$$\tau_p(\lambda(x), s_1, (n, m)) = \tag{A.29}$$

$$= \tau_e(s_1, (n, m)) + \left(\frac{\lambda(x)}{s_1} - 1 \right) \tau_e(s_1, (n, m)) = \tag{A.30}$$

$$= \frac{\lambda(x)}{s_1} \tau_e(s_1, (n, m)) = \tag{A.31}$$

$$= \frac{\lambda(x)}{s_1} \chi((n, m) \notin G) \max\{\varrho_{op}(n), \varrho_{ip}(m)\} K s_1 = \tag{A.32}$$

$$= \lambda(x) \chi((n, m) \notin G) \max\{\varrho_{op}(n), \varrho_{ip}(m)\} K = \tag{A.33}$$

$$= \frac{\lambda(x)}{s_2} \chi((n, m) \notin G) \max\{\varrho_{op}(n), \varrho_{ip}(m)\} K s_2 = \tag{A.34}$$

$$= \frac{\lambda(x)}{s_2} \tau_e(s_2, (n, m)) = \tag{A.35}$$

$$= \tau_e(s_2, (n, m)) + \left(\frac{\lambda(x)}{s_2} - 1 \right) \tau_e(s_2, (n, m)) = \tag{A.36}$$

$$= \tau_p(\lambda(x), s_2, (n, m)) \tag{A.37}$$

∎

Lemma A.4 (Performance characteristics of simultaneous transfer) Let $((h,\delta),q,s,l,r) \in \mathcal{D}_{st}$ be a simultaneous transfer. If $\forall x_1, x_2, ..., x_r \in \mathcal{B} : \lambda(x_1) = \lambda(x_2) = ... = \lambda(x_r) = l$, $q = (t_0, t_1, ..., t_m)$, and $k = \frac{l}{s}$ then $\forall i \in [1..r]$: the pipelined transfer time of x_i through path $\psi_i(q)$ with slice size s is the same and equals to:

$$\begin{aligned} &\sum_{j=1}^{m} \chi((t_{j-1},t_j) \notin h) \frac{rKs}{\min(r, \delta_{t_{j-1}}, \delta_{t_j})} + \\ &+ (k-1) \max_{j=1}^{m} \chi((t_{j-1},t_j) \notin h) \frac{rKs}{\min(r, \delta_{t_{j-1}}, \delta_{t_j})}, \end{aligned} \tag{A.38}$$

while latency is also the same and equals to:

$$\sum_{j=1}^{m-1} \chi((t_{j-1},t_j) \notin h) \frac{rKs}{\min(r, \delta_{t_{j-1}}, \delta_{t_j})} \tag{A.39}$$

Proof Let G be a structure that implements structure layout (h,δ) and $\forall i \in [1..r]$: $x_i = x_{i,1}x_{i,2}...x_{i,k}$, where $\forall u \in [1..k] : \lambda(x_{i,u}) = s$. Let $j \in [1..m], u \in [1..k]$. If

$$\forall i \in [1..r] : \text{transfer of } x_{i,u} \text{ from } \varphi_i(t_{j-1}) \text{ to } \varphi_i(t_j) \text{ starts at the same time,} \tag{A.40}$$

then it equals to:

$$\tau_e(\lambda(x_{i,u}), (\varphi_i(t_{j-1}), \varphi_i(t_j))) = \tag{A.41}$$

$$= \chi((\varphi_i(t_{j-1}), \varphi_i(t_j)) \notin G) \max\{\varrho_{op}(\varphi_i(t_{j-1}), \varrho_{ip}(\varphi_i(t_j)\} K\lambda(x_{i,u}) = \tag{A.42}$$

$$= \chi((t_{j-1},t_j) \notin h) \max\{\varrho_{op}(\varphi_i(t_{j-1}), \varrho_{ip}(\varphi_i(t_j)\} Ks = \tag{A.43}$$

$$= \chi((t_{j-1},t_j) \notin h) \max\left\{\max\left\{1, \frac{r}{\delta_{t_{j-1}}}\right\}, \max\left\{1, \frac{r}{\delta_{t_j}}\right\}\right\} Ks = \tag{A.44}$$

$$= \chi((t_{j-1},t_j) \notin h) \frac{rKs}{\min\{r, \delta_{t_{j-1}}, \delta_{t_j}\}} \tag{A.45}$$

Formula A.41 equals to formula A.42, based on definition 2.45. Formula A.42 equals to formula A.43, since G implements h (see statement 2.2 in definition 2.11) and $s = \lambda(x_{i,u})$. Formula A.43 equals to formula A.44, since on the one hand, if $(t_{j-1},t_j) \in h$ then both are equal to 0. On the other hand, if $(t_{j-1},t_j) \notin h$, than since G implements δ, ensuring that $\varphi_i(t_{j-1})$ is coupled with $\max\{1, \frac{r}{\delta_{t_{j-1}}}\}$

nodes of node type t_{j-1} and $\varphi_i(t_j)$ is coupled with $\max\{1, \frac{r}{\delta_{t_j}}\}$ nodes of node type t_j (see statement 2.4 in definition 2.13 and note that $\forall t \in \mathcal{T} : r$ is dividable by δ_t). Assumption A.40 ensures that each node of type t_{j-1} which $\varphi_i(t_{j-1})$ is coupled with transfers a byte array of size s and each node of type t_j which $\varphi_i(t_j)$ is coupled with recieves a byte array of size s. Since q is acyclic there is no other data transfer on these nodes. Therefore, $\varrho_{op}(\varphi_i(t_{j-1})) = \max\{1, \frac{r}{\delta_{t_{j-1}}}\}$ and $\varrho_{ip}(\varphi_i(t_j)) = \max\{1, \frac{r}{\delta_{t_j}}\}$. Formula A.44 equals to formula A.45, since r is never negative.

This also implies that if assumption A.40 is true, than $\forall i \in [1..r]$: transfer of $x_{i,u}$ from $\varphi_i(t_{j-1})$ to $\varphi_i(t_j)$ finishes at the same time. Definition 2.48 ensures that $\forall i \in [1..r]$: transfer of $x_{i,1}$ from $\varphi_i(t_0)$ to $\varphi_i(t_1)$ starts at the same time. By applying mathematical induction, it can be proven for any $j \in [1..m]$ and for any $u \in [1..k]$, that $\forall i \in [1..r]$: transfer of slice $x_{i,u}$ from $\varphi_i(t_{j-1})$ to $\varphi_i(t_j)$ starts at the same time. This means that line A.40 is always true.

Having τ_e for any $i \in [1..r], j \in [1..m]$, and $u \in [1..k]$, ε_p and τ_p (see definition 2.46) can be applied to determine latency and transfer time. ∎

Lemma A.5 (Simultaneous transfer of equivalent data flow cases)
Let (q_1, π_1) and (q_2, π_2) be equivalent data flow cases and let $((h_1, \delta), q_1, s, l, r)$, $((h_2, \delta), q_2, s, l, r) \in \mathcal{D}_{st}$. If h_1 and h_2 are elements of the instance layout sets that π_1 and π_2 define (respectively), then performance functions of the two simultaneous transfers are the same.

Proof First, it is showed that the lemma is true for directly equivalent data flow cases. Next, it is showed that it is true for all equivalent data flow cases.

Let (q_1, π_1) and (q_2, π_2) be directly equivalent data flow cases, where:

$$q_1 = (t_0, ..., t_{k-1}, t_k, t_{k+1}, ..., t_m), \tag{A.46}$$

$$q_2 = (t_0, ..., t_{k-1}, t_{k+1}, ..., t_m), \tag{A.47}$$

$$\pi_1(t_{k-1}, t_k) = true, \text{ and} \tag{A.48}$$

$$\pi_1(t_k, t_{k+1}) = false. \tag{A.49}$$

Based on definition 2.55:

$$\pi_2(t_{k-1}, t_{k+1}) = \pi_1(t_{k-1}, t_k) \wedge \pi_1(t_k, t_{k+1}) = \pi_1(t_k, t_{k+1}) \tag{A.50}$$

Based on definition 2.49, transfer time of simultaneous transfer $((h_1, \delta), q_1, s, l, r)$ equals to:

$$\tau_q((h_1, \delta), q_1, s, l, r) = \tag{A.51}$$

$$= \sum_{j \in [1..m]} \chi((t_{j-1}, t_j) \notin h_1) \frac{rKs}{\min(r, \delta_{t_{j-1}}, \delta_{t_j})} + \\ + \left(\frac{l}{s} - 1\right) \max_{j \in [1..m]} \chi((t_{j-1}, t_j) \notin h_1) \frac{rKs}{\min(r, \delta_{t_{j-1}}, \delta_{t_j})} = \tag{A.52}$$

$$= \sum_{j \in ([1..m]/k)} \chi((t_{j-1}, t_j) \notin h_1) \frac{rKs}{\min(r, \delta_{t_{j-1}}, \delta_{t_j})} + \\ + \left(\frac{l}{s} - 1\right) \max_{j \in ([1..m]/k)} \chi((t_{j-1}, t_j) \notin h_1) \frac{rKs}{\min(r, \delta_{t_{j-1}}, \delta_{t_j})} = \tag{A.53}$$

$$= \tau_q((h_2, \delta), q_2, s, l, r) \tag{A.54}$$

Formula A.51 equals to formula A.52 based on definition 2.49. Formula A.52 equals to formula A.53 because, based on condition A.49 and the fact that h_1 is an element of the instance layout set defined by π_1, $(t_{k-1}, t_k) \in h$ is true, $\chi((t_{j-1}, t_j) \notin h_1) = 0$. Formula A.53 equals to formula A.54 based on definition 2.49. Similarly, it can be showed that transfer times equal even in the case when $\pi_1(t_{k-1}, t_k) = false$ and $\pi_1(t_k, t_{k+1}) = true$.

Let (q_1, π_1) and (q_2, π_2) be equivalent data flow cases. Because definition 2.55 defines equivalence between data flow cases as a transitive, reflexive, symmetric closure of direct equivalence, there is a sequence of data flow cases that starts with (q_1, π_1), finishes with (q_2, π_2) and it is true for each neighbouring data flow cases in this sequence that they are directly equivalent meaning that their transfer times are the same. This implies that transfer times of (q_1, π_1) and (q_2, π_2) are the same as well. Similarly it can be proven that latency times are also the same. ■

Bibliography

[1] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[2] D.A. Reed. Grids, the TeraGrid and beyond. *Computer*, 36(1):62–68, 2003.

[3] TeraGrid. Online teragrid resource report. `http://www.teragrid.org/userinfo/hardware/resources.php`, 2009. [Online; accessed 26-Feb-2009].

[4] N. Geddes. The national grid service of the UK. *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 94, 2006.

[5] NGSGanglia. Ganglia - online ngs grid report. `http://ganglia.ngs.rl.ac.uk/`, 2009. [Online; accessed 26-Feb-2009].

[6] E. Laure, F. Hemmer, F. Prelz, S. Beco, S. Fisher, M. Livny, L. Guy, M. Barroso, P. Buncic, P. Kunszt, et al. Middleware for the next generation Grid infrastructure. *Proceedings of Computing in High Energy Physics and Nuclear Physics Conference*, 2004.

[7] EGEEProject. Egee project web page. `http://project.eu-egee.org`, 2009. [Online; accessed 26-Feb-2009].

[8] R. Niederberger and O. Mextorf. The DEISA project–Network operation and support–first experiences. *Computational Methods IN Science AND Technology*, 11(2):119–128, 2005.

[9] DEISA. Deisa architecture. `http://www.deisa.eu/deisa1/grid/architecture.php`, 2009. [Online; accessed 26-Feb-2009].

[10] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.

[11] T. Sandholm and J. Gawor. Globus Toolkit 3 Core–A Grid Service Container Framework, in: Globus Toolkit Core White Paper. `www.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf`. [Online; accessed 21-Oct-2009].

[12] E. Laure, SM Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, et al. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.

[13] B Schuller and FZ Julich. The unicore 6 grid middleware. 2007.

[14] M. Riedel, B. Schuller, D. Mallmann, R. Menday, A. Streit, B. Tweddell, M.S. Memon, A.S. Memon, B. Demuth, and T. Lippert. Web Services Interfaces and Open Standards Integration into the European UNICORE 6 Grid Middleware. *Proceedings of Middleware for Web Services Workshop at 11th International IEEE EDOC Conference*, pages 57–60, 2007.

[15] P. Asadzadeh, R. Buyya, C.L. Kei, D. Nayar, and S. Venugopal. Global grids and software toolkits: A study of four grid middleware technologies. *High-Performance Computing*, pages 431–458, 2005.

[16] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.

[17] P. Kacsuk, T. Kiss, and G. Sipos. Solving the grid interoperability problem by p-grade portal at workflow level. *Future Generation Computer Systems*, 24(7):744–751, 2008.

[18] T. Kiss, P. Kacsuk, G. Terstyanszky, and S. Winter. Workflow level interoperation of grid data resources. *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*, pages 194–201, 2008.

[19] Shantenu Jha, Hartmut Kaiser, Andre Merzky, and Ole Weidner. Grid interoperability at the application level using saga. *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 584–591, 2007.

[20] M. Riedel. Status of Grid Interoperation Now (GIN) Interoperability Activities. *OGF Workshop, e-Science 2007 Conference, Bangalore, India*, 2007.

[21] D.D. Roure, C. Goble, S. Aleksejevs, S. Bechhofer, J. Bhagat, D. Cruickshank, P. Fisher, N. Kollara, D. Michaelides, P. Missier, et al. The evolution of myexperiment. *Proceedings of the 2010 IEEE Sixth International Conference on e-Science*, pages 153–160, 2010.

[22] A. Goderis, U. Sattler, P. Lord, and C. Goble. Seven bottlenecks to workflow reuse and repurposing. *Proccedings of the Fourth International Semantic Web Conference*, 3792:323–337, 2005.

[23] D. Hollingsworth. The Workflow Reference Model. *Workflow Management Coalition*, 1995. Document Number TCOO-1003. Decument Status Issue 1.

[24] J. Yu and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3(3):171–200, 2005.

[25] Andreas Hoheisel. Grid Workflow Forum. `http://www.gridworkflow.org/snips/gridworkflow/space/Grid+Workflow`. [Online; accessed 4-Feb-2008].

[26] A. Hoheisel. Grid workflow execution service-dynamic and interactive execution and visualization of distributed workflows. *Proceedings of the Cracow Grid Workshop*, 2:13–24, 2006.

[27] S. Pellegrini, F. Giacomini, A. Ghiselli, and A. Hoheisel. Using GWorkflowDL for Middleware-Independent Modeling and Enactment of Workflows. *Proceedings of the CoreGRID Integration Workshop*, 2008.

[28] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record*, 34(3):44–49, 2005.

[29] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[30] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming scientific and distributed workflow with triana services. *Concurrency and Computation: Practice and Experience*, 18(10):1021–1037, 2006.

[31] S. Majithia et al. Triana: a graphical Web service composition and execution toolkit. *Proceedings of the IEEE International Conference on Web Services*, pages 514–521, 2004.

[32] Ian Taylor, Matthew Shields, and Ian Wang. *Grid resource management*, chapter Resource management for the Triana peer-to-peer services, pages 451–462. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[33] I. Taylor, M. Shields, and R. Philp. Gridoned: Peer to peer visualization using triana: A galaxy formation test case. *Proceedings of the UK eScience All Hands Meeting*, 2002.

[34] G. Allen, T. Goodale, T. Radke, M. Russell, E. Seidel, K. Davis, K.N. Dolkas, N.D. Doulamis, T. Kielmann, A. Merzky, et al. Enabling applications on the grid: a gridlab overview. *International Journal of High Performance Computing Applications*, 17(4):449, 2003.

[35] T. Oinn et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[36] T. Oinn, M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, et al. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.

[37] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):W729, 2006.

[38] P. Kacsuk, G. Dózsa, J. Kovács, R. Lovas, N. Podhorszki, Z. Balaton, and G. Gombás. P-GRADE: A Grid Programming Environment. *Journal of Grid Computing*, 1(2):171–197, 2003.

[39] P. Kacsuk, K. Karoczkai, G. Hermann, G. Sipos, and J. Kovács. WS-PGRADE: Supporting parameter sweep applications in workflows. *Proceedings on the Third Workshop on Workflows in Support of Large-Scale Science*, pages 1–10, 2008.

[40] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, and H.L. Truong. ASKALON: a tool set for cluster and Grid computing. *Concurrency and Computation: Practice and Experience*, 17(2):143–169, 2005.

[41] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler sys-

tem. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[42] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng. Overview of the Ptolemy Project. `http://www.ptolemy.eecs.berkeley.edu/publications/papers/03/overview/`, 2003. [Online; accessed 2-Mar-2009].

[43] E. Deelman. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[44] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and efficient workflow deployment of data-intensive applications on grids with MOTEUR. *International Journal of High Performance Computing Applications*, 22(3):347, 2008.

[45] K. Trichkov and E. Trichkova. Modeling and Execution of Web Service in Internet using Enhydra workflow platform. *Proceedings of the International Conference on Computer Systems and Technologies-CompSysTech*, 6, 2006.

[46] J. Frey. Condor DAGMan: Handling Inter-Job Dependencies. `http://www.bo.infn.it/calcolo/condor/dagman/`, 2002. [Online; accessed 5-Jun-2008].

[47] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, et al. Web services business process execution language version 2.0. *OASIS Standard*, 11, 2007.

[48] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, C. Goble, A. Wipat, P. Li, and T. Carver. Delivering web service coordination capability to users. *Proceedings of the 13th international World Wide Web conference on Alternate track papers and posters*, pages 438–439, 2004.

[49] E.A. Lee and S. Neuendorffer. MoML: A Modeling Markup Language in SML: Version 0.4. Technical report, 2000.

[50] C. Petri. *Communication with automata*. PhD thesis, PhD Dissertation of University of Bonn, 1962.

[51] R. Milner. *Communicating and mobile systems: the pi calculus*. Cambridge University Press, 1999.

[52] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. GridFTP: Protocol extensions to FTP for the Grid. *GWD-R (Recommendation)*, 2003.

[53] A. Rajasekar et al. Storage Resource Broker-Managing Distributed Data in a Grid. *Computer Society of India Journal, Special Issue on SAN*, 33(4):42–54, 2003.

[54] J.P. Baud, J. Casey, S. Lemaitre, C. Nicholson, D. Smith, and G. Stewart. Lcg data management: From edg to egee. *Proceedings of the UK eScience All Hands Meeting*, 2005.

[55] Amazon S3. Amazon simple storage service (amazon s3). `http://aws.amazon.com/s3/`, 2011. [Online; accessed 15-Jan-2011].

[56] J. Ellis, L. Ho, and M. Fisher. JDBC 3.0 Specification. `http://jcp.org/aboutJava/communityprocess/first/jsr054/index.html`. [Online; accessed 21-Feb-2009].

[57] M. Antonioletti et al. OGSA-DAI Usage Scenarios and Behaviour: Determining good practice. *Proceedings of the Third UK e-Science All Hands Meeting*, pages 818–823, 2004.

[58] M. Antonioletti et al. OGSA-DAI: Two Years On. *The Future of Grid Data Environments Workshop, The 10th Global Grid Forum*, 2004.

[59] WfMC. Workflow Management Coalition Terminology and Glossary. *Workflow Management Coalition*, 1999. Document Number WFMC-TC-1011. Document Status Issue 3.0.

[60] WFM-RG. Workflow Management Research Group Project Website. `http://forge.gridforum.org/sf/projects/wfm-rg`, 2008. [Online; accessed 23-Jan-2011].

[61] WfMC. Workflow Management Coalition Website. `http://www.wfmc.org`, 2008. [Online; accessed 13-Apr-2008].

[62] CppWfMS. CppWfMS on-line documentation. `http://wfms.forge.cnaf.infn.it/documentation/index.html`, 2008. [Online; accessed 27-May-2008].

[63] SHIWA. SHIWA project official web page. `http://shiwa-workflow.eu`. [Online; accessed 25-Jan-2011].

[64] WfMC. Workflow Management Coalition Workflow Standard - Interoperability Abstract Specification. *Workflow Management Coalition*, 1999.

[65] WMP van der Aalst and AHM ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[66] WMP van der Aalst, AHM ter Hofstede, B. Kiepuszewski, and AP Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[67] A. Brogi and R. Popescu. From BPEL processes to YAWL workflows. *Web Services and Formal Methods*, pages 107–122, 2006.

[68] S. Pellegrini, F. Giacomini, and A. Ghiselli. A practical approach for a workflow management system. *Proceedings of the CoreGRID Workshop*, 2007.

[69] A. Alqaoud, I. Taylor, and A. Jones. Scientific workflow interoperability framework. *International Journal of Business Process Integration and Management*, 5(1):93–105, 2010.

[70] A. Alqaoud, I. Taylor, and A. Jones. Publish/subscribe as a model for scientific workflow interoperability. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, pages 1–10. ACM, 2009.

[71] D. Box, LF Cabrera, C. Critchley, F. Curbera, D. Ferguson, S. Graham, D. Hull, G. Kakivaya, A. Lewis, B. Lovering, et al. Web services eventing. *World Wide Web Consortium (W3C) public draft, March*, 2006.

[72] B. Raoult, G. Aubert, and M. Gutierrez. The EU Funded SIMDAT Project: Components for Building the WMO Information System. *Technical Conference on the WMO Information System, Seoul, Korea*, 2006.

[73] M. Ghanem, N. Azam, and M. Boniface. Workflow Interoperability in Grid-based Systems. *Cracow Grid Workshop*, 2006.

[74] Z. Zhao, S. Booms, A. Belloum, C. de Laat, and B. Hertzberger. VLE-WFBus: A Scientific Workflow Bus for Multi e-Science Domains. *Proceedings of The Second IEEE International Conference on e-Science and Grid Computing*, pages 11–11, 2006.

[75] MySQL. MySQL 5.5 reference manual. `http://dev.mysql.com/doc/refman/5.5/en/`. [Online; accessed 23-Jan-2011].

[76] Oracle. Oracle Database 11g Release 2 reference. `http://download.oracle.com/docs/cd/E11882_01/server.112/e17110.pdf`. [Online; accessed 23-Jan-2011].

[77] PostgreSQL. PostgreSQL 7.3.21 Reference Manual. `http://www.postgresql.org/docs/7.3/static/reference.html`. [Online; accessed 23-Jan-2011].

[78] C. Mullins. *DB2 developer's guide*. Pearson Education - Sams, 5th edition, 2004.

[79] Microsoft. SQL Server Database Engine. `http://msdn.microsoft.com/en-us/library/ms187875.aspx`. [Online; accessed 23-Jan-2011].

[80] Xindice. Apache Xindice 1.1 Documentation. `http://xml.apache.org/xindice/1.1/index.html#Apache+Xindice+1.1+Documentation`. [Online; accessed 23-Jan-2011].

[81] eXist. eXist Open Source Native XML Database Documentation. `http://exist.sourceforge.net/documentation.html`. [Online; accessed 23-Jan-2011].

[82] A. Shoshani, P. Kunszt, H. Stockinger, K. Stockinger, E. Laure, J.P. Baud, J. Jensen, E. Knezo, S. Occhetti, O. Wynge, et al. Storage Resource Management: concepts, functionality, and interface specification. *Future of Grid Data Environments: a Global Grid Forum (GGF) Data Area Workshop*, 10:9–13, 2004.

[83] J. Postel. FTP: File transfer protocol specification. *Internet Engineering Task Force Request for Comments (RFC)*, 765, 1980.

[84] M.K. Smith, M. Barton, M. Bass, M. Branschofsky, G. McClellan, D. Stuve, R. Tansley, and J.H. Walker. DSpace: An open source dynamic digital repository. *D-Lib Magazine*, 9, 2003.

[85] S. Payette and C. Lagoze. Flexible and extensible digital object and repository architecture (FEDORA). *Research and Advanced Technology for Digital Libraries*, pages 517–517, 2009.

[86] Microsoft. *Microsoft ODBC 3.0: Software Development Kit and Programmer's Reference*. Microsoft Press Redmond, WA, 1997.

[87] T. Kukla, T. Kiss, G. Terstyanszky, and P. Kacsuk. Integrating open grid services architecture data access and integration with computational grid workflows. *Philosophical Transactions A of the Royal Socieaty*, 367:2521–2532, 2009.

[88] M. Antonioletti, N.P. Chue Hong, A.C. Hume, M. Jackson, K. Karasavvas, A. Krause, J.M. Schopf, M.P. Atkinson, B. Dobrzelecki, M. Illingworth, N. McDonnell, M. Parsons, and E. Theocharopoulos. OGSA-DAI 3.0–The Whats and the Whys. *Proceedings of the UK e-Science All Hands Meeting*, pages 158–165, 2008.

[89] M. Antonioletti, A. Krause, N.W. Paton, S. Laws, J. Melton, and D. Pearson. The WS-DAI family of specifications for web service data access and integration. *ACM SIGMOD Record*, 35(1):48–55, 2006.

[90] S. Fiore, M. Cafaro, A. Negro, S. Vadacca, G. Aloisio, R. Barbera, and E. Giorgio. GRelC DAS: A Grid-DB Access Service for gLite Based Production Grids. *Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 261–266, 2007.

[91] G. Aloisio, M. Cafaro, S. Fiore, and M. Mirto. A Split&Merge Data Management Architecture in a Grid Environment,. *Proceedings of the 19th IEEE International Symposium on Computer-Based Medical Systems (IEEE CBMS 2006), Salt Lake City Utah (USA)*, pages 739–744, 2006.

[92] G. Aloisio, M. Cafaro, S. Fiore, and M. Mirto. The GRelC library: a basic pillar in the grid relational catalog architecture. *Proceedings of the International Conference on Information Technology: Coding and Computing*, 1:372–376, 2004.

[93] G. Aloisio, M. Cafaro, and S. Fiore. Advanced delivery mechanisms in the GRelC project. *Proceedings of the 2nd workshop on Middleware for grid computing*, pages 69–74, 2004.

[94] B. Koblitz, N. Santos, and V. Pose. The AMGA Metadata Service. *Journal of Grid Computing*, 6(1):61–76, 2008.

[95] N. Santos and B. Koblitz. Distributed metadata with the AMGA metadata catalog. *Arxiv preprint cs/0604071*, 2006.

[96] A. Boloori, C. Cherubino, T. Calanducci, B. Koblitz, and S. Scifo. New developments in the glite amga metadata catalogue. Technical report, 2009.

[97] HadoopMapReduce. Hadoop MapReduce Online Documentation. `http://hadoop.apache.org/mapreduce/docs/current/`, 2011. [Online; accessed 9-Sep-2011].

[98] GRelCTeam. Grelc project. `http://grelc.unile.it`, 2010. [Online; accessed 16-Feb-2010].

[99] T. Delaitre et al. GEMLCA: Running Legacy Code Applications as Grid Services. *Journal of Grid Computing*, 3(1):75–90, 2005.

[100] T. Delaitre, A. Goyeneche, P. Kacsuk, T. Kiss, G.Z. Terstyanszky, and S.C. Winter. Gemlca: grid execution management for legacy code architecture design. *Euromicro Conference, 2004. Proceedings. 30th*, pages 477 – 483, aug.-3 sept. 2004.

[101] A. Abdelnur and S. Hepper. JSR 168: Portlet specification. *Java Specification Requests, Java Community Process, Sun Microsystems and IBM*, 15, 2003.

[102] A. Yang, J. Linn, and D. Quadrato. Developing integrated Web and database applications using JAVA applets and JDBC drivers. *Proceedings of the 29th SIGCSE technical symposium on Computer science education*, pages 302–306, 1998.

[103] Tavernateam. Ogsa-dai wsi 2.2 taverna 1.4 proof-of-concept. `http://www.ogsadai.org.uk/downloads/ogsadai-wsi-2.2-taverna-1.4`. [Online; accessed 9-Jun-2008].

[104] D. Kodeboyina and B. Plale. Experiences with OGSA-DAI: Portlet Access and Benchmark. *Global Grid Forum Workshop on Designing and Building Grid Services*, 2003.

[105] X. Yang et al. Integration of Existing Grid Tools in Sakai VRE. *International Workshop on Collaborative Virtual Research Environments, GCC*, 2006.

[106] X. Yang et al. A Web Portal for the National Grid Service. *Proceedings of the UK e-Science All Hands Meeting*, pages 1156–1162, 2005.

[107] Tamas Kiss and Tamas Kukla. High-level user interface for accessing database resources on the grid. *Proceedings of the 7th International Conference on Distributed and Parallel Systems*, pages 155–163, 2008.

[108] Tamas Kiss, Tamas Kukla, and Gabor Terstyanszky. Towards Grid data interoperation: OGSA-DAI data resources in computational Grid workflows. *Proceedings of the CoreGRID Integration Workshop*, 2008.

[109] Tamas Kukla. Integrating the OGSA-DAI to the P-GRADE portal. Master's thesis, 2007.

[110] P. Kacsuk, A. Goyeneche, T. Delaitre, T. Kiss, Z. Farkas, and T. Boczko. High-level grid application environment to use legacy codes as ogsa grid services. *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 428–435, 2004.

[111] M. Ghanem, N. Azam, M. Boniface, and J. Ferris. Grid-Enabled Workflows for Industrial Product Design. *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing*, 2006.

[112] M. Ghanem, N. Azam, and M. Boniface. Workflow Interoperability in Grid-based Systems-Presentation. *Cracow Grid Workshop*, 2006.

[113] V. Curcin, M. Ghanem, P. Wendel, and Y. Guo. Heterogeneous Workflows in Scientific Workflow Systems. *Proceedings of the 2nd Int. Workshop on Workflow Systems in e-Science in conjunction with the Int. Conference on Computational Science*, pages 27–30, 2007.

[114] M. Surridge, S. Taylor, D. De Roure, and E. Zaluska. Experiences with GRIA: Industrial Applications on a Web Services Grid. *e-Science and Grid Computing, First International Conference on*, pages 98–105, 2005.

[115] Z. Zhao, A. Belloum, C. De Laat, P. Adriaans, and B. Hertzberger. Using Jade agent framework to prototype an e-Science workflow bus. *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid*, pages 655–660, 2007.

[116] Z. Zhao, A. Belloum, C. de Laat, P. Adriaans, and B. Hertzberger. Distributed execution of aggregated multi domain workflows using an agent framework. *Proceedings of the IEEE Congress on Services*, pages 183–190, 2007.