A Framework for Grid-Enabling Scientific Workflow Systems

Architecture and application case studies on interoperability and heterogeneity in support for Grid workflow automation.

Nabeel Adeem AZAM

A thesis submitted for the degree of
Doctor of Philosophy

Department of Computing
School of Informatics
University of Bradford

2010
Abstract

Nabeel Azam

A Framework for Grid-Enabling Scientific Workflow Systems

Keywords: Grid, workflow, service-oriented paradigm, interoperability

Since the early 2000s, Service Oriented Architectures (SOAs) have played a key role in the development of complex applications within a virtual organization (VO) context. Grids and workflows have emerged as vital technologies for addressing the (SOA) paradigm. Given the variety of Grid middleware, scientific workflow systems and Grid workflows available, bringing the two technologies together in a flexible, reusable and generalized way has been largely overlooked, particularly from a scientific end user perspective. The lack of domain focus in this area has led to a slow uptake of Grid technologies.

This thesis aims to design a framework for Grid-enabling workflows, which identifies the essential technological components, how these components fit together in layered architecture and the interactions between them. To produce such a framework, this thesis first investigates the definition of a Grid-workflow architecture and mapping Grid functionality to workflow nodes, focusing on striking a balance between performance, usability and the Grid functionality supported. Next, it presents an examination of framework extensions for supporting various forms of Grid heterogeneity, essential for
VO based collaboration. Given the complex nature of Grid technologies, the work presented here investigates abstracting Grid based workflows through high-level definitions and resolution using semantic technologies. Finally, this thesis presents a way to resolves abstract Grid workflows using semantic technologies and intelligent, autonomous agents.

The frameworks presented in this thesis are tested and evaluated within the context of domain-based case studies defined in the SIMDAT, BRIDGE and ARGUGRID EU funded research projects.
I would first like to thank my supervisor Dr. Geyong Min for getting me started down the research path, and for his support, patience and encouragement over the years. I would also like to thank Pr. Yike Guo for giving me the opportunity to work for InforSense.

A special thank you to Dr. Moustafa Ghanem for taking me under his wing. I would never have been able to complete this thesis without his guidance and motivation. Another special thank you to Ginger Frost. I could never have reached this point without her help and belief in me.

Cheers to all my colleagues/friends at InforSense, past and present, for putting up with me over the past few years and letting me pick their brains. I would particularly like to thank Ben Coiffe, Vasa Curcin, Nikolaos Giannadakis, Francois Gilardoni, Li Guo, Dimitris Kalaitzopoulos, Mauro Mellino, Alex Michie, Michelle Osmond, Jameel Syed and Patrick Wendel.

A big thank you to all the EU project partners for their collaboration and cooperation. Much appreciated!

Finally, thank you to my parents and my brother Umayr for their love and support. Oh, and all my friends and family for constantly chirping “are you done yet?” Thanks everyone!
Publications


# Table of Contents

ABSTRACT ..................................................................................................................... I

ACKNOWLEDGEMENTS .............................................................................................. III

PUBLICATIONS ........................................................................................................ IV

TABLE OF CONTENTS ................................................................................................. V

LIST OF FIGURES ......................................................................................................... XI

LIST OF TABLES .......................................................................................................... XVI

LIST OF ABBREVIATIONS ........................................................................................... XVII

CHAPTER 1 INTRODUCTION ....................................................................................... 1

1.1 BENEFITS ASSOCIATED WITH SOA ADOPTION .............................................. 2

1.2 PROBLEM AND CHALLENGES ............................................................................ 3

1.3 ORIGINAL CONTRIBUTIONS .............................................................................. 6

1.4 THESIS STRUCTURE ............................................................................................ 9

CHAPTER 2 BACKGROUND ......................................................................................... 12

2.1 SERVICE ORIENTED ARCHITECTURES ............................................................ 13

2.1.1 The High-Level SOA Model .......................................................................... 13

2.1.2 The Extended SOA Model ............................................................................ 15

2.1.3 SOA Layer Classification ............................................................................. 17

2.1.4 Addressing SOAs and Conclusions ............................................................... 20

2.2 GRID AND WEB SERVICES ............................................................................... 21

2.2.1 Web Services .................................................................................................. 22

2.2.2 Grid Computing ............................................................................................. 26

2.2.3 Examples of Grid Middleware ....................................................................... 33
4.4.6  Adaptive API Approach Implementation .......................................................... 171
4.4.7  BRIDGE Case Study: Updated DockFlow ....................................................... 173

4.5  EFFECT OF SERVICE LOCALITY ON DockFLOW DESIGN ........................................ 175
4.5.1  Local and Remote Services ............................................................................. 176
4.5.2  Implementation of DockFlow Pre-Processing Steps ......................................... 177

4.6  HETEROGENEITY PERFORMANCE EVALUATION .................................................. 179
4.6.1  Comparing the Gateway and Adaptive API Approaches .................................... 179
4.6.2  Comparing Pre-Processing Operations ............................................................. 184

4.7  SUMMARY AND CONCLUSIONS ............................................................................. 186

CHAPTER 5 SUPPORTING HIGH-LEVEL WORKFLOW DEVELOPMENT .......... 189
5.1  SUPPORTING ABSTRACTION ................................................................................. 190
5.1.1  Mapping Abstract Workflows to Concrete Workflows ......................................... 194
5.1.2  Service Mapping Categories ............................................................................. 195

5.2  ABSTRACTION-BASED GRID-WORKFLOW ARCHITECTURE .............................. 200
5.2.1  Abstract Handler Definition .............................................................................. 201
5.2.2  Defining the Abstract Node .............................................................................. 203
5.2.3  Abstract Support Interfaces .............................................................................. 204
5.2.4  Registry ............................................................................................................. 205
5.2.5  Interactions ....................................................................................................... 206
5.2.6  Implementation for Supporting Abstraction ...................................................... 208
5.2.7  Case Study: Abstracting DockFlow ................................................................. 211

5.3  RUNTIME SERVICE SELECTION ......................................................................... 215

5.4  EXTENDING ABSTRACTION TO SUPPORT RUNTIME SELECTION .................. 217
5.4.1  Grid and Semantic Broker Handler ................................................................... 219
5.4.2  Nodes ................................................................................................................. 220
5.4.3  Semantic Broker Interface .............................................................................. 222
5.4.4  Runtime Service Selection (Broker) Interactions ............................................. 223
5.4.5  Runtime Service Implementation ..................................................................... 224
CHAPTER 6 MOVING TOWARDS AUTOMATED SERVICE COMPOSITION
THROUGH THE USE OF INTELLIGENT AGENTS ........................................236

6.1 REQUIREMENTS FOR AUTOMATION ..................................................237

6.2 AN EXTENDED ARCHITECTURE TO SUPPORT DISCOVERY AGENCIES ..........239

6.2.1 Conceptual View ............................................................................239

6.2.2 Semantic Annotations and Ontologies .............................................242

6.2.3 Service Registries ...........................................................................243

6.2.4 Brokering Systems and Translation Tools ........................................244

6.2.5 SLAs .............................................................................................245

6.3 SEMANTIC LEVELS OF ABSTRACTION .............................................246

6.4 SERVICE COMPOSITION MODELS ..................................................249

6.4.1 Resolving Two Mismatching Services Located on the Same Grid Server ....251

6.4.2 Resolving Two Mismatching Services Located on Different Grid Servers ....254

6.5 MODELS FOR SERVICE SELECTION ...............................................257

6.5.1 Models for Attributes ....................................................................257

6.5.2 Workflow Arrangement Models ......................................................259

6.5.3 Using Models .................................................................................264

6.5.4 Workflow Arrangement Models Summary .......................................265

6.6 SEMANTIC WORKFLOW COMPOSITION SUPPORT .............................266

6.6.1 Semantic-Based Grid-Workflow Architecture ...................................266

6.6.2 Semantic Node Definition ...............................................................268

6.6.3 Semantic Support Interfaces ............................................................268

6.6.4 Semantic Interactions .....................................................................270

6.6.5 Implementation of Semantic Components .......................................272

6.6.6 Semantic Handler Implementation ..................................................278

6.7 ARGUGRID CASE STUDIES: EARTH OBSERVATION .........................282
List of Figures

Figure 2.1: The high-level SOA model ................................................................. 14
Figure 2.2: Extended SOA model with the inclusion of Discovery Agencies ........ 16
Figure 2.3: Classification of the SOA layers and their association with Service Requestors and Providers ........................................................................ 18
Figure 2.4: An example of a web services protocol stack .................................. 24
Figure 2.5: Conversion of Web and Grid service technologies and standards ...... 28
Figure 2.6: Relationship between OGSA and WSRF for the development of Grid systems ...................................................................................................... 29
Figure 2.7: Architectural arrangement of Grid profiles ....................................... 33
Figure 2.8: Grid middleware architecture .......................................................... 38
Figure 2.9: Anatomy of a Workflow Node .......................................................... 40
Figure 2.10: WfMC architecture for workflow management systems ................. 43
Figure 2.11: Workflow user roles and their relationships .................................... 46
Figure 2.12: Layered workflow architecture ...................................................... 59
Figure 2.13: Layout of InforSense Workflow Builder for constructing workflows .... 64
Figure 2.14: Combined workflow and Grid layered architecture in accordance with the SOA paradigm ...................................................................................... 66
Figure 3.1: Layered Grid-workflow architecture with Grid handler layer definition ..... 71
Figure 3.2: Overview of the Single Node, R-DTE and DTE-M node mappings for accessing Grid functionality ............................................................. 74
Figure 3.3: Grid Handler definition for the Single Node Mapping ....................... 76
Figure 3.4: Pseudo code example of the Single Node’s process method for invoking a Grid service ........................................................................................ 80
Figure 3.5: Single Node interactions for Administration and Configuration operations required before workflow authoring and execution ............................. 83
Figure 3.6: Component interactions for executing a Grid service using the Single Node approach .................................................................84

Figure 3.7: InforSense portal based configuration page for the GRIA handler ........87

Figure 3.8: Single GRIA node used within a workflow displaying interfaces for allocating resources ........................................................................90

Figure 3.9: Grid-Enabled WEKA workflow demonstrating seamless connectivity between Grid node and other workflow nodes .............................................93

Figure 3.10: Automotive Workflow for Crash Simulation demonstrating Grid service invocation and composition.................................................................95

Figure 3.11: Grid Handler and component definitions for the RDTE Node Mapping....98

Figure 3.12: Pseudo code for Resource Allocation node design, producing a resource allocation model ................................................................................102

Figure 3.13: Pseudo code design for Upload data node that takes a resource allocation model and produces resource allocation and data reference models ................................................................................103

Figure 3.14: Pseudo code for Execute job node design that takes and produces data and resource allocation reference models .................................................104

Figure 3.15: Pseudo code for Download data node that copies data from the Grid server to the Workflow server and produces a file reference .........................105

Figure 3.16: Component Interactions for Administration and Configuration.........107

Figure 3.17: R-DTE interactions for retrieving a resource allocation..................108

Figure 3.18: Interactions for executing a R-DTE Upload Data node ...................109

Figure 3.19: R-DTE interactions for invoking a job service ...............................111

Figure 3.20: Interactions for downloading the results from a service ................112

Figure 3.21: Node Parameters for Resource Allocation Node and interface for determining a resource allocation at authoring time ..............................................114

Figure 3.22: Updated Crash Simulation Workflow using the RDTE Approach. More nodes are required for each service execution, but the performance and usability are improved ................................................................................116
Figure 4.5: Schematic of DockFlow concept ................................................................. 161
Figure 4.6: Complete DockFlow scenario using Gateway Approach ......................... 162
Figure 4.7: Layered architecture for supporting the Adaptive API approach ............. 165
Figure 4.8: Adaptive API Grid Handler ........................................................................ 166
Figure 4.9: Interactions for Adaptive API approach ................................................. 171
Figure 4.10: Revised DockFlow using the Adaptive API approach ......................... 174
Figure 4.11: DOCK pre-processing steps using the local installation/bespoke components method ................................................................. 178
Figure 4.12: Data transfer times for Gateway and Adaptive API approaches ............ 181
Figure 4.13: Execution times for direct, Adaptive API and Gateway approaches ...... 182
Figure 4.14: Job execution times for the Gateway and Adaptive API approaches as the number of molecules used increases ......................................................... 183
Figure 4.15: AutoGrid (AutoDock pre-processing tool) execution times ................. 185
Figure 4.16: AutoGrid execution times with data transfer ....................................... 186
Figure 5.1: Encapsulation of DTE-based Grid workflows ........................................ 192
Figure 5.2: Mapping a three node workflow to particular implementations of available services ..................................................................................................... 193
Figure 5.3: Extended layered architecture to support abstraction ......................... 201
Figure 5.4: Abstract Handler Definition .................................................................... 202
Figure 5.5: Interactions for abstract workflow resolution ........................................ 207
Figure 5.6: Abstract workflow interface for retrieving matching workflow fragments ........................................................................................................ 210
Figure 5.7: AutoDock workflow fragment requiring manual shimming ................... 212
Figure 5.8: Layered architecture for automated runtime service selection .............. 218
Figure 5.9: Extended Grid Handler for runtime service selection .......................... 219
Figure 5.10: Interactions for registry access .............................................................. 223
Figure 5.11: Architecture of NEC Semantic Broker ................................................. 226
Figure 5.12: Grid flow with Semantic Broker-based service selection node .......... 228
Figure 5.13: Layer separation in the IX-Odus workflow ......................................... 230
Figure 6.1: Conceptual view of the proposed environment for automatic service composition .......................................................... 241

Figure 6.2: Levels of abstraction required for semantically describing workflows .... 248

Figure 6.3: Using shimming to bridge the gap between services ......................... 250

Figure 6.4: Two abstract descriptions resolving to two mismatching services ....... 251

Figure 6.5: Changing one service or both services in order to resolve mismatches . 252

Figure 6.6: Adding shims in order to resolve a mismatch between two services ..... 253

Figure 6.7: Example of two mismatch services located on different Grid servers .... 254

Figure 6.8: The affect of service location on the decision making process .......... 255

Figure 6.9: Factoring the location of a shim service to resolve service mismatching .................................................................................................................. 256

Figure 6.10: Identifying the arrangements within a complex workflow .............. 265

Figure 6.11: Extensions, applied to a Grid-workflow layered architecture, to support semantic workflow composition .......................................................................................................................... 267

Figure 6.12: Interactions for submitting an abstract workflow to the broker .......... 271

Figure 6.13: Interactions for retrieving and rendering a concrete workflow .......... 272

Figure 6.14: Internal structure of Argubroker ................................................... 276

Figure 6.15: Semantic node with associated annotations. The annotations are entered as WSMO documents and parsed for clearer viewing ................................................. 279

Figure 6.16: Returned concrete workflow with an associated SLA. The workflow is not executable until the SLA is accepted and submitted to the Grid server ........ 282

Figure 6.17: Returned workflow for getImage. No inputs so only a getImage and Download Data node are returned by ArguBroker ................................................. 288

Figure 6.18: A semantic workflow for retrieving an image and applying a pollution map ........................................................................................................................................... 290

Figure 6.19: Concrete workflow for pollutionMap scenario ............................... 290

Figure 6.20: Concrete workflow with shims returned for the pollutionMap scenario. 292

Figure 6.21: Comparisons between sequential, parallel models and actual results .. 299

Figure 6.22: Comparison between loop models and actual results ..................... 300
List of Tables

Table 2.1: Workflow system comparison and overview ........................................52
Table 2.2: Comparison of workflow system features ...........................................55
Table 3.1: Comparison of workflow and native data upload times .......................137
Table 3.2: Comparison of workflow and native data download times ....................138
Table 3.3: Comparison of workflow and native job execution times ....................139
Table 3.4: Execution times for each node mapping with different file sizes and
number of services, each running on a different GRIA server ..........................141
Table 3.5: Execution times for each node mapping with different file sizes and number of services, each running on the same GRIA server ..........................142
Table 4.1: Comparison of data transfer times of Gateway and Adaptive API
approaches ........................................................................................................180
Table 4.2: Job Execution times for direct GOS access, Adaptive API and Gateway
approaches ........................................................................................................182
Table 4.3: Job execution times for the Gateway and Adaptive API approaches as the
number of molecules used increases ..................................................................183
Table 4.4: AutoGrid (AutoDock pre-processing tool) execution times ..................185
Table 4.5: AutoGrid execution times with data transfer ......................................186
Table 5.1: Properties associated with service mapping categories .......................197
Table 6.1: Breakdown of workflow execution times for a single service accessed
using the DTE-M mapping ..................................................................................260
Table 6.2: Response times for ArguBroker comments .......................................295
Table 6.3: Comparisons between sequential, parallel composition models and actual
results ..................................................................................................................298
Table 6.4: Comparison between loop models and actual results ..........................300
List of Abbreviations

AGWL Abstract Grid Workflow Language
ARGUGRID ARGUmentation as a foundation for the semantic GRID
API Application Programming Interface
AXIS Apache eXtensible Interaction System
BLASTN Basic Local Alignment Search Tool Nucleotide
BPEL Business Process Execution Language
BPML Business Process Markup Language
BRIDGE Bilateral Research and Industrial Development enhancing and Integrating Grid Enabled technologies
CaSAPI Credulous and Sceptical Argumentation (Prolog Implementation)
CAD Computer Aided Design
CNGrid China National Grid
CoG Java Commodity Grid Kit
CPU Central Processing Unit
DAG Directed Acyclic Graph
DAGMan Directed Acyclic Graph Manager
DMML Data Mining Markup Language
DTE Data Transfer and job Execution
DTE-M Data Transfer, job Execution and Metadata
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE-DI</td>
<td>Functionality Equivalent with Different Interfaces</td>
</tr>
<tr>
<td>FE-EI</td>
<td>Functionality Equivalent with Exact Interfaces</td>
</tr>
<tr>
<td>FS-DI</td>
<td>Functionality Similar with Different Interfaces</td>
</tr>
<tr>
<td>FS-EI</td>
<td>Functionality Similar with Exact Interfaces</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>FTPS</td>
<td>File Transfer Protocol Secure</td>
</tr>
<tr>
<td>GAT</td>
<td>Grid Access Toolkit</td>
</tr>
<tr>
<td>GGF</td>
<td>Global Grid Forum</td>
</tr>
<tr>
<td>GOLEM</td>
<td>Generalized OntoLogical Environments for Multi-agent systems</td>
</tr>
<tr>
<td>GOS</td>
<td>Grid-Oriented Storage</td>
</tr>
<tr>
<td>GRAM</td>
<td>Grid Resource Allocation and Management</td>
</tr>
<tr>
<td>GRIA</td>
<td>Grid Resources for Industrial Applications</td>
</tr>
<tr>
<td>GridARM</td>
<td>Askalon’s Grid Resource Management</td>
</tr>
<tr>
<td>GriddLeS</td>
<td>Grid Enabling Legacy Software</td>
</tr>
<tr>
<td>GridSAM</td>
<td>Grid Submission And Monitoring web service</td>
</tr>
<tr>
<td>Grip</td>
<td>Grid process API</td>
</tr>
<tr>
<td>GT</td>
<td>Globus Toolkit</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>GWFE</td>
<td>Gridbus WorkFlow Engine</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext Transfer Protocol Secure</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Computing</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>JSDL</td>
<td>Job Service Description Language</td>
</tr>
<tr>
<td>JSP</td>
<td>JavaServer Pages</td>
</tr>
<tr>
<td>KNIME</td>
<td>KoNstanz Information MinEr</td>
</tr>
<tr>
<td>MARGO</td>
<td>Multiattribute ARGumentation framework for Opinion explanation</td>
</tr>
<tr>
<td>MSAP</td>
<td>Master Sequence Analysis Pipeline</td>
</tr>
<tr>
<td>MTOM</td>
<td>Message Transmission Optimization Mechanism</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>OGSA</td>
<td>Open Grid Services Architecture</td>
</tr>
<tr>
<td>OGSI</td>
<td>Open Grid Service Infrastructure</td>
</tr>
<tr>
<td>OMII-UK</td>
<td>Open Middleware Infrastructure Institute-United Kingdom</td>
</tr>
<tr>
<td>OWL</td>
<td>Ontology Web Language</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer To Peer</td>
</tr>
<tr>
<td>PLATON ++</td>
<td>P2P Load Adjusting Tree Overlay Networks</td>
</tr>
<tr>
<td>PROSOCOS</td>
<td>PROgramming SOcieties of ComputeeS</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>R-DTE</td>
<td>Resource allocation, Data Transfer and job Execution</td>
</tr>
<tr>
<td>REST</td>
<td>REpresenational State Transfer</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SIMDAT</td>
<td>SIMulating DATa grids</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>SwA</td>
<td>SOAP with Attachments</td>
</tr>
<tr>
<td>TUAM</td>
<td>Tool for Universal Annotation and Mediation</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery Integration</td>
</tr>
<tr>
<td>ULB</td>
<td>Université Libre de Bruxelles</td>
</tr>
<tr>
<td>URI</td>
<td>Unique Reference Identification</td>
</tr>
<tr>
<td>VO</td>
<td>Virtual Organisation</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WEKA</td>
<td>Waikato Environment for Knowledge Analysis</td>
</tr>
<tr>
<td>WIMC</td>
<td>Workflow Management Coalition</td>
</tr>
<tr>
<td>WS</td>
<td>Web Service</td>
</tr>
<tr>
<td>WS-I</td>
<td>Web Services Interoperability Organization</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSFL</td>
<td>Web Service Flow Language</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
<tr>
<td>WSRF</td>
<td>Web Service Resource Framework</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XScufl</td>
<td>Simple Conceptual Unified Flow Language</td>
</tr>
<tr>
<td>xWFL</td>
<td>XML-based WorkFlow Language</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

Both scientific workflow systems and Grid middleware play an important part in a Service Oriented Architecture (SOA) for allowing organizations and individuals to collaborate in an “ecosystem” environment, essentially forming a Virtual Organization (VO) [46][86]. The SOA paradigm builds around the notion of a service, a self-contained component or activity that is a seen as a black box with known interfaces. The SOA model consists of two main components, a Service Provider, who is responsible for publishing and hosting services and a Service Consumer who accesses these services, composing them together to form applications to solve complex domain problems. Grids address the Service Provider perspective by providing a means for deploying and publishing applications as Grid services, providing the security and management mechanisms for accessing them and for coordinating multiple resources, thus maintaining the trust required within a VO [45][100][7][102]. Workflows address the consumer perspective through the provision of tools that allow users to discovery, access, compose, and invoke services in a set of logical ordered steps [20].

Given the functionality provided by Grids, the Grid service selection, discovery and composition processes becomes inherently more complex
Introduction

when compared to other forms of services (i.e. web services, local tools), especially within a VO context, where services are hosted by various organisations and require secure access as well as policies governing the execution. An extended SOA model addresses these complexities using the concept of discovery agencies to aid the aforementioned Grid service processes. Discovery agencies provide a means of storing and indexing services and their descriptions allowing users to make informed decisions about which services are best suited to addressing their requirements. Discovery agencies can range from simple registries to intelligent systems that can perform the selection, negotiation and composition on behalf of the end user.

1.1 Benefits Associated with SOA adoption

Adopting a SOA-based approach for the construction of applications has a number of distinct advantages over the more “fortress” based approaches where applications are built in-house in a static, monolithic environment. Services can be reused and act as building blocks to construct further, coarser-grained services and applications. Within a SOA, it is possible to substitute services integrated into applications with alternative service implementations as long as the required behaviour stays relatively the same or produces the same results [77]. This interchangeable service property associated with SOAs allows for the optimization of applications, preventing
applications from becoming obsolete or redundant. This optimization can occur in terms of performance or cost, re-direction of requests to an alternate service if network connections or machines fail [99] or keeping applications up-to-date as service implementations evolve. SOAs provide better fault tolerance compared to distributed object systems through the handling of service boundaries explicitly rather than emphasizing distribution transparency, the latter leading to difficulties in determining failure modes [35]. The location transparency property of SOAs, dynamic service lookup and binding allows organisations to move code and services to different machines and providers without breaking applications. This property also provides scalability through load balancing and distribution of service requests without the knowledge of the service client [99]. Furthermore, as described in [74][66], SOAs are known to increase performance and business revenue through increased convenience and agility.

1.2 Problem and Challenges

A wide variety of Grid middleware and scientific workflow systems are available for building complex applications within a collaboration setting, with each implementation suited for particular situations and addressing specific domain areas. The workflow composition process becomes even more difficult given the nature of Grid services, and the extra security and management functions associated with them. The SOA paradigm lacks a set
of standardized integration methodologies for bringing the two technologies together, especially from a less technical domain perspective.

The main challenge lies in defining a flexible and extensible integration framework based on the SOA models. This framework is required to strike a balance between usability for scientific users with largely domain expertise and little technical knowledge, whilst leveraging the benefits provided through the utilization of Grid functionality. The sub challenges associated with such a framework include:

- **How to select and compose Grid services through workflows:** A set of workflow nodes, the atomic unit used to represent tasks in a workflow, are required for handling Grid functionality. These nodes must allow for seamless integration with other workflow nodes and strike a balance between the functionality supported and usability. Describing these Grid-based nodes at too low a level reveals technical details that can render them difficult for domain experts to use. Describing the nodes at too high a level can lead to loss of supported Grid functionality and reduce composition flexibility. The performance of workflows composed of Grid based nodes requires consideration, particularly optimization through tasks such as the minimization of workflow overheads associated with accessing Grid services.
Introduction

- **How to address support for heterogeneity:** Essentially a range of scientific workflow systems and Grid middleware implementations are available for building applications in a SOA context. An effective framework must allow for the loose coupling of Grid middleware and workflow systems, thus allowing the specific implementations of each of the technologies to be interchanged. Supporting heterogeneity introduces further problems that require addressing, such as minimizing loss of Grid functionality, stabilizing performance and maintaining usability.

- **How to provide workflow abstractions for supporting the extended SOA model:** Introducing discovery agencies to alleviate some of the Grid service selection and composition complexities requires the ability to define workflows at a higher, more abstract level. To support this higher level of abstraction a framework must describe extensions for workflow systems to support abstract nodes for composing abstract workflows and resolving abstract nodes to concrete implementations of services.

- **How to utilize intelligent systems in building Grid-based workflows:** Resolving abstract workflows manually, i.e. the end user has to choose the services to use, can be a long and laborious process considering the number of variables involved. Using
intelligent automated systems to perform this operation on behalf of the workflow author is one method of addressing this issue. An effective framework must be able to interface with these discovery agencies and provide the relevant information to them in order to resolve abstract workflows. This is not a simple task, given that composition, as well as individual service matching, requires consideration and thus requires the use of workflow composition and execution models.

The work presented in this thesis addresses these challenges using a number of scientific case studies. The closer these case studies are to real life scenarios, the more effective the evaluation and validity of the overall framework. Case studies defined in a number of Grid based EU funded projects are used, namely SIMDAT [95], BRIDGE [14] and ARGUGRID [4][30].

1.3 Original Contributions

The main contributions of this thesis are as follows:

- The design and definition of a generic framework for addressing the SOA paradigm based on Grid enabling scientific workflow systems.
Such a framework is required to focus on addressing the challenges from a scientific end user’s perspective and requires:

- a layered architecture that considers the complete SOA model and an identification of the components within the architecture;

- an investigation of existing Grid workflow node mappings that identifies their strengths and weaknesses, as well as an evaluation of their usability and performance;

- determination of the interactions between components in the layered architecture when invoking Grid services, and the operations associated with them.

- Extending the framework to support Grid heterogeneity by providing workflow support for accessing and interoperating between Grid middleware, maintaining the node considerations described previously. These extensions include:

  - updating the architecture to support two forms of identified Grid interoperability;

  - identifying the new interactions between components for supporting heterogeneity;
• Abstraction of workflow and nodes for the application of higher-level descriptions to support an extended SOA model based on Discovery agencies. Achieving this abstraction requires the identification of:

  o ways of mapping abstract, high-level workflow definitions to concrete, executable workflows;

  o the criteria and properties for mapping abstract to concrete workflows;

  o extensions to a Grid-workflow framework for supporting authoring time and automated runtime service selection based on the use of semantic technologies.

• Finally, a full Grid-workflow framework for supporting automated workflow resolution via autonomous agents. Such a framework consists of:

  o a revised layered architecture, interactions and node definitions;
Introduction

- workflow composition models for intelligent service selection and composition;
- identification and analysis of workflow patterns;
- service selection models based on QoS parameters.

To summarize, the overall aim is to provide a flexible framework for addressing different SOA-based models that allows for the interchanging of Workflow and Grid technology implementations, whilst maintaining a level of usability for domain experts.

1.4 Thesis Structure

The structure for the rest of this thesis is as follows:

**Chapter 2**: Presents a detailed overview of the SOA paradigm and its usage in building complex applications. Grid and workflow are two technologies within the SOA paradigm, and Chapter 2 reviews both technologies, focusing on their architectural structure for supporting integration with other technologies.
Chapter 3: Identifies an initial framework that specifies the mapping of Grid functionality to workflow nodes, the architectural components involved and the interactions between them. These node mappings are tested and evaluated in terms of performance as well as usability, for specific scenarios across the automotive, knowledge services and pharmaceutical domains.

Chapter 4: Extends the work presented in Chapter 3, investigating support for two possible methods of Grid heterogeneity via the framework defined in the previous chapter. These framework extensions are tested within the context of a collaborative protein-docking scenario that also identifies the importance of service deployment locality. This chapter evaluates both Grid heterogeneity and service locality.

Chapter 5: Investigates how workflow abstraction can simplify the Grid workflow construction processes defined in the previous chapters. Also identifies and applies to pharmaceutical scenarios the criteria for abstraction, authoring time service resolution, and automated runtime resolution through semantic technologies.

Chapter 6: While the previous chapter focuses solely on service selection, Chapter 6 investigates workflow models and composition patterns, particularly how they affect the Grid service selection process. This chapter also identifies how the framework supports intelligent agents to automate the
Introduction

Grid service selection and composition processes. Earth Observation scenarios test the work described in this chapter.

Chapter 7: Concludes this thesis and identifies possible opportunities for future work in this field.
Chapter 2
Background

Service Oriented Architectures (SOAs) provide a flexible environment for building complex applications through cross-organisation collaboration. Building these complex applications entail the controlled co-ordination and execution of other low-level applications hosted at different sites or by different organisations as services. Web and Grid service technologies in the context of SOA models provide the hosting, management, authentication and policy mechanisms required to govern access to these remote services and computational resources. Scientific Workflow systems sit at the other end of the SOA paradigm, providing users with a means to construct further domain-specific applications through the invocation and orchestration of remote services.

This chapter reviews Grid and scientific workflow technologies, identifying the key components in each of their architectures in order to determine how to combine the two technologies together in a SOA context. The flaws and advantages of existing Grid workflow systems, as well as the failings and advances of previous studies in this area, identify a need for a framework that addresses Grid-enabling scientific workflow systems detailed herein as an SOA based architectural solution.
2.1 Service Oriented Architectures

A Service Oriented Architecture (SOA) is a structured set of loosely coupled and interoperable autonomous components [92]. The SOA paradigm has recently emerged as a way of addressing the changing trends in building complex, high performance applications and achieving global collaboration between businesses and academia. SOAs adopt an ecosystem view [86] for building distributed applications, combining a range of vendors, individuals, services, resources and environments across different ownership domains, thus forming a Virtual Organisation (VO) [46]. Collaborating within the context of VOs requires the developers of applications to address the quality of service (QoS) described in terms of resource usage and management, security semantics, fail over and functionality across a range of heterogeneous resources [45]. Multi-organisational Grid environments aim to tackle the QoS challenges inherited through the formation of VOs and used within the context of the SOA paradigm.

2.1.1 The High-Level SOA Model

A range of SOA models and definitions has arisen over the past few years focusing on different domains (e.g. industry, business, scientific), environments and interoperating standards (e.g. Web Services and Grid) with varying degrees of complexity and specifics. This section describes the architecture, concepts and terms associated with the SOA model at a high-
level. Figure 2.1 shows the components within the conceptual SOA model and the basic interactions associated with them.

![Service Requestor](image1) ![Service Provider](image2)

**Figure 2.1: The high-level SOA model**

At the heart of the SOA paradigm is the concept of a service. A service is a self-contained software component providing functionality and/or a representation of an activity that participates in realizing one or more capabilities [105][77]. Services have well defined interfaces and generally appear as encapsulated black boxes to those accessing them. Services can be atomic self-contained applications, composed of other services or dependent on other services and resources such as databases [98].

Service Requestors (sometimes referred to as Service Consumers) are entities that can represent individuals, organisations or systems that access, interact with and invoke services. The Service Requestor interacts with the services via message exchanges through client interfaces or neighbouring services. Service Requestors are usually bound to an agreement or contract in order to access services [92].
The third component in the high-level service model is the Service Provider. The Service provider is responsible for hosting one or more services and/or resources, i.e. storage and HPC (High Performance Computing) resources. The Service provider publishes service descriptions and endpoints (i.e. URIs or addresses) required for accessing each hosted service. Note that service access usually occurs remotely within a VO consisting of Service Requestors and Providers. Thus agreements are entered, and governed, by policies and contracts in order to determine access rights and levels of service using security mechanisms such as encryption, authentication and Service Level Agreements (SLAs).

### 2.1.2 The Extended SOA Model

One drawback of the high-level SOA model is that it does not address service discovery. In many instances, the Service Requestor may not know the location or details of a required service. To address this issue Discovery Agencies can be added to the high-level SOA model as shown in Figure 2.2, which is adapted from [19][89][5].
At their simplest, Discovery Agencies can be represented as service registries which advertise available services provided by the Service Provider. The Service Provider publishes within a registry for service discovery, a searchable set of service endpoints and descriptions that can range from free text to ontological terms and semantic annotations. When building applications, the Service Requestor can query this registry to find services that match their requirements. For each query, the registry (where possible) provides a list of matching services thus allowing Service Requestors to begin interactions with the Service Provider. As well as service endpoints, Service Registries can also hold information regarding
both service functionality and QoS properties that govern each service’s execution, allowing Service Providers to make well-informed service selection decisions as covered in Chapters 5 and 6. Note that within the SOA model, there is not a strict one-to-one mapping between service registries and Service Providers and a single registry can be associated with multiple Service Providers.

Regarding scenarios that are more complex, Discovery Agencies can represent service brokers, which are trusted third parties that force Service Providers and Requestors to adhere to data/usage privacy regulations and best practices, thus maintaining trust and guaranteeing levels of service. Service brokers add value to service registries through the provision of extended annotations that include information about reliability, trustworthiness and consistency [89]. Like service brokers, autonomous agents can also act as mediators between Service Requestors and Providers. Agents can also perform the service searching, decision making and contract negotiation operations on behalf of the Service Requestor as addressed in Chapter 6. Champion in [19] provides more information on further SOA patterns.

2.1.3 SOA Layer Classification

Like SOA Models, SOA layer classification comes in many different flavours, with each classification suited for varying levels of scenario complexity,
different domains and different environments. For example, a model such as the one described in [5] is specifically designed for addressing business processes whereas the model in [38] is tailored towards the eScience domains. Figure 2.3 provides a generalised model that amalgamates the commonality between a range of models provided in [5][83][106].

![Figure 2.3: Classification of the SOA layers and their association with Service Requestors and Providers](image)

The bottom three layers of the model are associated with the Service Provider, while the top two are associated with the Service Requestor. Taking a bottom-up approach, the layers are as follows:

- **Operational Systems Layer:** The Operational Systems layer, sometimes referred to as the data layer, houses data stores and infrastructure programs. Databases or file spaces represent these stores that hold the data items required by the services or Service
Provider. Infrastructure programs include operating systems, computational resources and other tertiary support applications.

- **Enterprise Components Layer:** The Enterprise Components layer primarily consists of applications wrapped and accessed as services. These applications interact with the components in the Operations Systems Layer, which govern the application execution and manage the transfer of data to/from the application. Note that composing applications at this level allows for the formation of coarser-grained, more complex applications, which in turn are accessible as a single service.

- **Services Layer:** The Services Layer is responsible for the wrapping and deployment of applications as well as for providing access to resources located at the Operation Systems Layer via Grid and Web service technologies. Endpoints (e.g. URIs) and interface descriptions (such as WSDL files), are exposed at this level for service discovery, bindings and message exchanges through technologies/protocols such as REST and SOAP.

- **Process Layer:** The Process layer (also referred to as the Orchestration or Business Logic layer) involves handling the invocation and co-ordination of service executions, essentially forming a workflow. Service Requestors can write workflows using specialised
workflow systems or functional programming/scripting languages, such as the Java based GROOVY language. In some SOA layer classifications, there is no differentiation between the service orchestration occurring at the Services and Process layer. Emig in [36] justifies a separation between the co-ordination processes, with the service level performed from a technical viewpoint and the process level performed from a domain perspective. Process-level workflows are covered in depth later in this chapter.

- **Consumer Layer:** The top layer, in some instances known as the Presentation or Enterprise layer, presents application front-ends, interfaces and access points allowing individuals to interact with the SOA stack. This level can include Grid/Web Service clients, workflow clients and web portal applications for non-technical domain experts.

The Security and QoS vertices that traverse the SOA layer stack (shown in Figure 2.3) represent the authentication and agreements required to interact with the different layers of the stack and the conditions that govern the service executions respectively.

### 2.1.4 Addressing SOAs and Conclusions

A number of technologies are available for supporting the layer separations and service models that make up SOAs. Grid and web service technologies
support the notion of a remote service, which is paramount to the SOA paradigm, and address the Service Provider perspective by providing security standards, access interfaces, resource/data management and, in the case of Grid computing, resource scheduling/distribution. From a Service Requestor viewpoint, workflow technologies address the Process and Consumer layers presented in the SOA layer classification and provide a means of accessing and composing Grid or web services.

In Figure 2.3, workflow systems address the top two layers (Service Requestor perspective). The Grid Middleware supports the bottom three layers (Service Provider perspective). The remainder of this chapter will investigate these technologies in more detail, and their usage within the SOA paradigm, in order to identify ways of bridging the communication gap between Service Providers and Service Requestors.

### 2.2 Grid and Web Services

Web and Grid service technologies allow computers to communicate in a distributed, platform-independent environment. Both technologies are therefore core components of the SOA paradigm. This section describes these technologies in more detail, investigating how they relate to each other and support the SOA Service Provider viewpoint.
2.2.1 Web Services

The World Wide Web Consortium (W3C) defines Web Services as “a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards” [58]. Most Web services are usually implemented around three specifications:

- **SOAP (Simple Object Access Protocol):** SOAP [115] is an XML (eXtensible Mark-up Language) [117] based message exchange protocol, for exchanging structured and typed information between systems [115].

- **WSDL (Web Service Description Language):** WSDL [22] is an XML formatted document for defining the interfaces for a service including endpoints, operations supported, messages defining the data to be transmitted and data type definitions.

- **UDDI (Universal Description Discovery Integration):** UDDI [85] is an XML based registry for Web services, managing information about implementations, metadata and Service Providers.
While a combination of these three specifications may be the common standard for implementing Web services, other methods such as REST [40], do exist and try to address the shortcomings of the specifications. REST (Representational State Transfer) is an architectural style definition for web services as opposed to a set of specifications or toolkits. Both Web service implementation methods have their advantages and disadvantages making them suited for particular situations as described in [67].

A range of standards and specifications are proposed by varying consortiums, primarily W3C [114] OASIS [87] and WS-I [108]. These specifications are designed to be either interoperable or extensible to provide alternatives for the three core protocols (SOAP, WSDL and UDDI) identified previously. Figure 2.4 below shows one possible representation of the web services stack based on [62], identifying how the protocols are categorized and related.
Background

Interoperability
Includes: Basic, Attachments, Simple SOAP Binding, Basic Security, REL Token, SAML Token and Reliable Asynchronous Messaging Profiles. All are part of the WS-I specifications.

Business Processes
Includes the languages BPEL4WS, BPML, XPDL and CDL4WS

Metadata
Includes WSDL, UDDI, WS-Policy specifications, WS-Discovery and WS-MetadataExchange

Resource
Includes WSRF specifications from OASIS and WS-Transfer, RRSHB from W3C

Messaging
Includes the languages BPEL4WS, BPML, XPDL and CDL4WS

Security
Includes: WS-Security profiles for SOAP messaging, Kerberos binding, SAML Token and X.509 Certificate Token from OASIS. Other specifications include WS-Federation, WS-Trust and WS-SecureConversation

Figure 2.4: An example of a web services protocol stack

Note that there is no standard classification of protocols or layered specification stack for Web Services. [11][113] provide other web service stacks definitions. Based on Figure 2.4, the main specification categories are as follows:

- **Interoperability**: Provides specifications by WS-I that attempts to improve interoperability between protocols in the Web services stack. Note that these are not standards, but rather a set of best practices and guidelines.
Background

- **Business Process**: Defines specifications for coordination and choreographing the execution of Web services including languages such as BPEL and BPML.

- **Metadata**: Standards for describing services and their interfaces, as well as identifying how these services can be stored, indexed and searched. Specifications such as WS-Policy [116] define a framework for describing domain specific functionalities, security settings and QoS constraints.

- **Messaging**: Includes protocols such as SOAP and specifications for the exchange of data and transport mechanisms such as tokens (WS-Addressing) and notifications (WS-Eventing).

- **Resource**: Specifications that attempt to define a generic and open framework for modelling and accessing stateful resources using web services [63]. Unlike stateless services, stateful services keep track of interactions and their states. The WSRF (Web Service Resource Framework) set of specifications [54] addresses these issues.

- **Security**: Communication protocols that define Web Service support for authentication when accessing services, encryption and message integrity/confidentiality.
The vast array of specifications have come about to support computer interoperability—an essential part of the SOA paradigm. Given the flexibility of SOAs, various combinations of standards can be used to implement web services, depending on the nature and complexity of the SOA and VO environment. Specifications, such as those related to security and metadata, have seen a convergence between Web and Grid standards.

### 2.2.2 Grid Computing

Grid computing has had many definitions since its conception in the early nineties. In 1998, Foster and Kesselman formally described a computational Grid as “a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities” [44]. This definition, along with many others, likened computational Grids to that of traditional power Grids: sharing and distributing resources when needed. These definitions were updated to account for the concept of VOs, and the focus became the negotiation between parties over resources, access rights and data privacy [46]. Foster in [41] updated the definition of Grid based on a three point checklist stating that a Grid is a system that “coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver non trivial qualities of service.” The three points are as follows:
Background

- **Coordination:** Coordinating resources within different domains found either across organizations or within the same organization, addressing security, policy, payment and membership.

- **Standard Protocols:** The definition requires Grid protocols, policies and interfaces to be open, general purpose and standard. As a result, dynamic establishment of resource-sharing arrangements between parties is possible, and systems are compatible and interoperable.

- **Quality of Service:** A Grid coordinates resources usage to deliver varying QoS; the utility of the combined system is consequently significantly greater than that of the sum of its parts [41].

**Convergence of Grid and Web Service Standards**

Initially both Web and Grid services allowed individuals or organizations to access services remotely, facilitating cross-organization collaboration. Web services standards provided a platform independent solution for making remote procedure calls and the passing of XML based messages. This particular method of invocation had its setbacks; the XML messaging format was unable to handle binary data, the lack of security specifications in the standards and the stateless nature of Web services made them unsuitable for computationally intense jobs that require the aforementioned features. Grid services, on the other hand, did address these features, but their complexity was unsuitable for small, quick tasks and the lack of well-defined
standards lead to separate vendor-specific interfaces and communication protocols [122]. Web services became more like the Grid computing paradigm as its set of standards expanded to include specifications, such as SwA (SOAP with Attachments) and MTOM (Message Transmission Optimization Mechanism) for handling binary data, WS-Security for authentication and WSRF for supporting state. Grid standardization recommendations, such as OGSI (Open Grid Service Infrastructure), prompted a convergence between the two technologies as shown in Figure 2.5 modified from [16].

![Conversion of Web and Grid service technologies and standards](image)

**Figure 2.5: Conversion of Web and Grid service technologies and standards**

This convergence eventually led to the notion of Grid services extending and complementing Web services rather than designating them as separate technologies [122][56][90], thus moving towards the OGSA (Open Grid Services Architecture) standard. The OGSA standard is based on Web
Background

service protocols such as SOAP, WSDL, WS-I for insuring interoperability and WSRF, the latter which replaced OGSI definitions for state support [31].

OGSA

The OGSA standard is based on the SOA paradigm and defines a set of core capabilities and behaviors for resolving key Grid concerns such as establishing identities, negotiating authentication, expressing and negotiating policies, service discovery, negotiation and monitoring of SLAs and communications within a VO setting [47]. Figure 2.6 adapted from [96] shows how OGSA and WSRF are related in the development of Grid systems.

![Figure 2.6: Relationship between OGSA and WSRF for the development of Grid systems](image-url)
Background

OGSA packages Grid functionality into seven groups, each addressed by a set of profiles as defined in [47]. As these are recommendations, not all the groups (or all parts of the groups) are required when implementing a Grid infrastructure:

- **Infrastructure Services**: A set of required foundation interfaces supported by all the other services. The choice of Web services for the underlying infrastructure promotes the SOA ideology with XML as the de facto language for descriptions, SOAP as the message exchange protocol and WSRF for addressing state. Security services for ensuring authentication, data encryption and service access authorization are also required using protocols such as WS-Security and WS-Agreement for policies.

- **Execution Management Services**: Three classes of services that define execution handling. Resource Services, provides a container for managing resource allocations, data storage and executables. Job Services are responsible for the preparation, invocation and management of an instance of a unit of “work” or running a task. Selection services plan and schedule the service invocation and resources to meet agreed service levels.
Background

- **Data Services**: These services manage the access, updating and transfer of data sources. The types of data resources supported by this architecture include flat files, streams and catalogues.

- **Heterogeneity and Extensibility**: Provides OGSA functionality to non-OGSA based architectures. This profile also supports the use of API wrappers to enable third parties to access OGSA functionality; however, there is no standard or scope defined for these APIs.

- **Functional Capabilities**: Profiles the functional capabilities provided by OGSA data services. Since the use of different subsets depends on the requirements of the SOA, these services are separate from the other groups. Functional capabilities include securing data transfer between locations and applying individual access rights to data items, storage management, resource and service configuration, metadata and accessing provenance information.

- **Resource Management Services**: Covers three types of management: physical resources (machines), OGSA Grid resources accessed through service interfaces (resource allocation, job submission and monitoring) and the OGSA infrastructure.

- **Security Services**: Facilitates the enforcement of security-related policies within a VO. These capabilities include authentication and
identity mapping for verifying user identity (using mechanisms such as username/passwords or keys and certificates), authorization to determine access rights to data resources and services, privacy to hide identification information from other users where necessary.

- **Self-Management Services:** Used in conjunction with Security Services to automate particular tasks that would usually be performed by human resources. These services include handling SLAs and policies in order to maintain QoS.

- **Information Services:** Provides a means to access and modify information about applications, resources and data. These services apply to both static services, such as application metadata and service discovery and to dynamic data, such as runtime log information and job monitoring.

Figure 2.7 shows an architectural arrangement of components based on the profiles listed and the architecture defined in [47]. These components are removable, combinable or interchangeable depending on the requirements and design of the Grid system. Infrastructure services are at the bottom of the architecture as they provide the basis for all the other services. The heterogeneity and extensibility layer provides an interface for other systems to access the Grid functionality and thus sits at the top of the architecture.
2.2.3 Examples of Grid Middleware

The Grid middleware examples described here support Foster’s [41] definition of Grid. This definition excludes systems such as Sun Grid Engine, due to its centralized control, and Condor, which uses specialized protocols rather than open standards.

**Globus Toolkit 4**

The fourth version of the Globus Toolkit (GT4) developed by the GGF, the same group that produces the OGSA standards, consists of modular packages that can be interchanged depending on the configuration, as described in the OGSA definitions. GT4 provides containers for wrapping Java, C and Python applications, allowing for the deployment of programs written in these languages as services. Based on and extending WSRF standards, these containers provide the security and management mechanisms required to interact with these applications. GT4 publishes metadata about each service using an index service [23], a registry similar to UDDI. This metadata, described as WSRF resource properties, comes in an
XML document format [85]. GT4 provides a set of libraries and tools for the file transfer and data access mechanisms based on the GridFTP specification, allowing for interoperability with FTP based clients and servers [43]. GT4 uses the Globus Resource Allocation and Management (GRAM) service to handle the request and allotment of resources, as well as job execution management. A GRAM service allows clients to describe the resource allocation, reference data items and pass arguments. GT4 uses GRAM as a management tool for deploying and instantiating a service, controlling the resource and providing fail-safe measures [43]. GT4 uses X.509 public key credentials as the basis for security mechanisms, although further security measures can be implemented [42].

From a client perspective, GT4 provides APIs and libraries for building client-side applications to perform resource management, service discovery, data transfer/management and job invocation/monitoring. APIs and libraries for Java, C and Python are available. As GT4 is built around WS-I recommendations, all services have a common interoperable interface regardless of the language used to deploy the application, and thus a Java based client can be used to access Python wrapped application and so forth. The Java Commodity Grid (CoG) Kit provides a Java based implementation of the Globus protocols and functions [70] at a higher level, specifically for building client based applications.
GRIA

GRIA (Grid Resources for Industrial Applications) [64][100] is a free and open source infrastructure from IT Innovation. GRIA provides several software packages: the Basic Application Services and Service Provider Management packages address the Service Provider perspective and GRIA Client and Client Management packages both address the Service Requestor perspective. Basic Application Services enable the encapsulation and deployment of applications, access to data storage and management of resources. Service Provider Management offers security and policy mechanisms such as SLA support and billing/accounting services. The GRIA Client package applications interface with provider-side services for example data, service and job interaction as well as management services. The Client Management package offers additional Service Requestor based management such as control and monitoring of access and usage [64].

A number of OGSA recommendations form the base for GRIA. It is built upon the Apache AXIS web service middleware which is compliant with the WS-I Basic Profile, conforms to the WS-Addressing, which is in turn part of the WSRF specifications and a number of WS-* specifications, such as WS-Security, WS-Addressing and WS-Notification, for authentication/authorization, support for state and usage monitoring respectively [63].
GRIA has its own wrapper script definition to encapsulate and deploy legacy applications. These scripts define the interfaces and arguments that need to be supplied from clients and can also be used to provide additional functionality, e.g. wrap more than one application and co-ordinate the transfer of data between them or perform functions such as data archiving.

GRIA provides three services: a Job service for searching, accessing, invoking and monitoring instances of application executions; a Data service for handling file transfer, indexing and managing data items; and Resource allocation for negotiating resources and application metadata in XML format. The GRIA Client package also provides APIs and Libraries for the development of Java based applications for accessing GRIA functionality.

**GOS**

GOS is a suite of Grid software that supports the CNGrid (China National Grid) running environment [21]. GOS provides applications for job and batch job submissions, file services for file transfer based on HTTP and FTP protocols, deployment of new services, accounting services and a messaging service for communication between GOS hosts. GOS is based on a range of WS-* standards, such as WS-I and WS-Security. While file transfer currently relies on HTTP and FTP (as well as SwA), the GOS roadmap includes future updates to the file transfer implementation in accordance with GridFTP. GridSAM, produced by the OMII-UK group [81] and based around the JSDL (Job Service Description Language) specifications [3], handles Job submission. Due to its design, it is possible to
host GOS in a variety of environments such as Apache Tomcat with AXIS, GT4 and OMII.

GOS also provides a client side AP, the Grid Process API (Grip), for constructing applications and invoking its services. Grip is based on five basic operations key to Grid services; create, bind, invoke, control and close.

2.2.4 Addressing SOAs Using Grid/Web Services and Conclusions

Both Grid and Web Services technologies support the Service Provider perspective within a SOA. Due to the convergence of the latest Grid and Web service standards, both technologies are complementary, and can be used together to address the needs of a SOA. The latest Grid perspective is from a collaborative VO perspective rather than that of resource management and scheduling optimization. Based on the OGSA specifications and Grid middleware study [2], a commonality exists between the functions available and the groupings of services for performing basic, client level actions are identifiable. As shown in Figure 2.8, these basic actions fall into the categories resource allocation, data services for data transfer and management, job services for invocation/monitoring, access management (security), information services for indexing/searching for services and application metadata.
Grid systems provide a general API that supports client side applications for invocation and management of applications and data items. However, each API is vendor and implementation specific. Although researchers have explored the definition of common APIs, none have been adopted as a standard, particularly due to the benefits of the open specifications that allow great flexibility.

2.3 Workflow Systems

A workflow is a set of steps, coupled with the information flow between them, required for executing real world processes. Each step is defined by
activities, which either software tools or humans execute. Work, in the form of data or jobs, passes through the different steps in a specified order. Each of these steps may invoke a service, and these services can either be a hard-coded or a third party application accessed locally or remotely. The ability to access and coordinate remote services advocates workflows as a high-level, user centric approach to developing distributed applications in general. In the service oriented computing context, this ability makes workflows theoretically suitable for building Grid service-based applications [20].

2.3.1 Workflow Graphs

Generally, scientific workflows are directed graphs consisting of nodes and edges. Nodes can represent activities or monolithic tasks such as the invocation of local processing operations or remote service calls, while edges (also referred to as connections or links) between each node represent the flow of data. Directed graphs associated with workflows are often acyclic, however, special cases justify cyclic graphs.

Nodes and Edges

Each node can be configured, or inputs can be entered via a set of parameters. Nodes may also have input and output ports for receiving and transmitting data. Each edge has a source and a target, with the source
being the output of a node and the target being the input of another node.

Figure 2.9 shows the anatomy of a workflow node adapted from [25].

![Anatomy of a Workflow Node]

**Figure 2.9: Anatomy of a Workflow Node**

Inputs to a node can be provided via ports that take data produced by other nodes, and parameters, which are usually manually entered by the workflow author and generally represent simple data types such as Strings, Integers and Booleans (although some workflow systems support more complex data types). All nodes must produce at least one output, but do not necessarily need an input – a condition usually indicative of a starting node).

**Types of Flow**

The flow of information represented in a workflow essentially falls under two categories: data flow and control flow. Data flows are usually DAGs (Directed Acyclic Graphs) and are concerned with the passing of information or data between nodes. Data flows can operate based on both pull and push
models. In a pull model, the user determines the final node or endpoint of a workflow with the edges indicating dependencies, i.e. determining the subsequent nodes to execute for ensuring the execution of the final selected node. Each node depends on the ones that precedes it, and waits for data to arrive at its inputs before execution. A push model, executes the first node or nodes in a workflow followed by all subsequent nodes until the ends of all the workflow branches are satisfied [26].

Control flow operations employ a push model and are concerned with the transfer of tokens that govern the co-ordination of execution rather than the flow of data. Control flows allow for control operations such as conditionals (IF/ELSE, SWITCH) and looping (WHILE, DO/WHILE, FOR/EACH).

Due to the differences in modes of operation, some workflow systems separate the data and control flow layers, with control flows used to govern the co-ordination of data flow executions.

**Metadata**

The concept of metadata describes the inputs, outputs and parameters of each node as well as the data structures that are passed between the nodes, e.g. a tabular data structure can be defined by its columns and the type of values those columns hold. Metadata validates nodes, edges and workflows to ensure that valid data structures are passed to inputs or valid values are entered as parameters, and to ensure a meaningful chaining of nodes.
without having to perform executions of the workflow and to ensure that passing of valid data structures to inputs or entry of valid values into parameters. For example if the metadata defined for a parameter limits the type of values entered to integers only and within a specific range, any entries beyond that range, or of a different type, can raise a flag and in some cases prevent the node and subsequent nodes from executing until the problem is resolved. The metadata for ports can determine whether an input is compulsory, e.g. requires data to be passed to it in order for the node to be executed, or optional, as well as the type or types of data that are accepted by an input or produced by an output.

2.3.2 Workflow System Architecture

Workflows can be used to solve a range of problems across a number of domains, with certain workflow systems specialising in a particular field, e.g. although both Taverna and Kepler are scientific workflow systems, the former focuses on life sciences and the latter on ecology and geology [55] [6]. On the other hand, the Oracle BPEL Process Manager is a workflow enactment engine (or server) and client interface for the BPEL language that focuses on co-ordinating web services for business processes [38]. Even though workflow systems come in many flavours, they generally follow the same architectural model.
Workflow Reference Model

The Workflow Management Coalition (WfMC), a world-wide organisation consisting of workflow end users, consultants, developers, university groups and researchers [112], defines a global, high-level reference model for the architecture of workflow management systems [60] (Figure 2.10).

Service Providers use clients for constructing process definitions, or otherwise known as generating workflows. These clients are usually front-end visual GUIs that have representations of the execution steps and ways of defining the work between steps. When a workflow is completed and invoked, the client submits the process definition to the workflow enactment engine, which executes the workflow. The Workflow Client Application interface provides a set of APIs for client applications to request services.
Background

from the workflow engine to monitor and control processes and work. The invoked application interface is a definition of APIs allowing the workflow engine to invoke a range of remote services, such as web and Grid services through common software. The workflow interoperability interface is a definition of standard to support interworking with other workflow enactment engines. Although workflow systems can come in a standalone configuration, the WfMC architecture supports multiple clients connecting to a single workflow enactment engine.

User Roles

Building workflows for addressing complex scientific problems usually involves a range of personnel whose specific skill sets, access rights and specialities make them suited for particular functions and stages in the workflow lifecycle. These personnel can fall in one or more of the following roles:

- **Administrator**: Given that administrators generally deal with the low-level technical details, they are responsible for determining the configuration and security settings that govern workflow execution, as well as user management. These responsibilities include determining which workflow functions and nodes are available to each set of workflow authors, the paths and proxy settings for accessing third party applications or spaces, and low-level configuration such as memory and space allocation.
• **Workflow Author:** Workflow authors are responsible for building workflows based on requirements provided by domain experts. A workflow author’s skill set lies between administrators and domain experts as they must have technical expertise and understanding of the workflow technology, as well as a general grasp of the domain in order to successfully build solutions. Complex workflows may be too technical and difficult for domain experts to use, therefore most workflow systems provide authors with a mechanism for encapsulating the inner workings of the workflow, and deploying the workflow as a service, via a service interface such as a portal, or at a higher, non-editable higher level. These deployments encapsulate the workings of the workflow, hiding the details and only revealing the parameters and ports chosen by the workflow author for the domain expert to interact with, providing a clear, easy to use interface.

• **Domain Expert:** Domain experts can access workflows via the process definition client, but they are usually concerned with the application of the workflow rather than the building process, and their skill sets center on their qualification/specialty areas more so than on their technical knowledge. These specialists normally access workflows through other interfaces, such as web portals, command line based interfaces, web and Grid services.
Figure 2.11 shows the relationships between the user roles and the interfaces they interact with in the workflow lifecycle.

2.3.3 Workflow Systems

A wide variety of available workflow systems address the SOA models. These include scientific workflows designed for specific scientific domains, business process workflows focusing on process management, and Grid workflows specially designed to orchestrate services running on particular Grid middleware. The following systems are examples of tools for building workflows across a range of application domains.

InforSense

InforSense is a commercially available data and process driven workflow
system based on the Discovery Net [1][29][94] platform and is suited mainly for life sciences, but also provides support for business processes and other domains [9][51][79]. The InforSense system follows a client/server architecture with a web based portal interface for administration purposes and execution of deployed workflows. InforSense describes workflows using DMML, an XML based language. The system consists of a J2EE platform, with clients based on Java Web Start technology. It is possible to couple the client and server in a standalone configuration, or to separate them allowing multiple clients to connect to a single server.

**Taverna/Freefluo**

The Freefluo workflow enactment engine from IT Innovation supports two XML based workflow languages; WSFL from IBM, and XScufl developed as part of the Taverna project [61][97]. Both Taverna and Freefluo are Java-based. The Taverna component originated from the myGrid project [61] and supports the life sciences community, such as biology, chemistry and medical imaging [55], particularly focusing on the invocation and coordination of web services. Taverna can orchestrate a combination of local and remote resources, composing and linking them in a data flow.

**Triana**

Triana is a Java-based graphical environment developed by the University of Cardiff that allows for the assembly of programs using a list of building blocks [75][104]. The Triana system consists of two layers, a Triana GUI and a
Triana Service. Workflows produced by the client generate an XML-based, WSFL-like TaskGraph. The Triana Service consists of three main components, specifically a client, server and command process server, with the latter acting as an enactment engine. It is viable to network multiple Triana systems together with a single active command process.

**Kepler**

Kepler is based on the mature Ptolemy II system for analysing and modelling scientific data, developed in Java at the University of California, Berkley [65][71]. A server/client architecture allows workflows to be created in the GUI interface or submitted via command line to the server. Kepler uses the concept of Actors and Directors with Actors representing nodes or atomic steps and Directors responsible for co-ordinating the execution of Actors. Kepler specialises in aiding the construction of workflows in the ecology and geology domains [6].

**KNIME**

The Konstanz Information Miner (KNIME) is a data flow platform developed by the Chair of Bioinformatics and Information Mining at the University of Konstanz [13]. KNIME is Java-based and has a graphical workflow client, provided as a standalone configuration, or as an Eclipse plug-in for supporting node development. Note that the enactment engine and client come coupled as a single application rather than a separate client-server configuration.
DAGMan and Pegasus

The Directed Acyclic Graph Manager (DAGMan) is a meta-scheduler for Condor [24]. DAGMan provides a means for coordinating services by specifying the order of execution and the dependencies between the jobs scheduled. DAGMan does not have a visual client for building workflows, with workflows specified in textual format instead. Packages such as Graphviz are available for rendering DAGMan workflows visually, however, these are not interactive or used to create workflows, but rather to check the progress of a workflow during execution. Pegasus [34] provides a means of producing abstract workflows by referencing catalogued data items and services. These catalogue links are resolved at runtime to actual data items and services. Coupling Pegasus with DAGMan [34][32] provides a visual front end for Condor jobs and for defining Kepler abstract workflows [76].

GWFE

The Gridbus Workflow Engine (GWFE) [119] provides an XML-based workflow language xWFL that allows users to define tasks and their dependencies. In GWFE, the definition of abstract workflows requires manual specification of the applications to run, and the engine identifies Grid Service Providers at runtime for the production of concrete workflows. GWFE uses the Gridbus Broker to interface to multiple Grid middleware, including different versions of the Globus toolkit, and scheduling systems such as Sun Grid Engine. GWFE does not have a visual editor but does provide a visual graph at runtime indicating the current progress of tasks.
Background

Askalon

The Askalon [39] workflow system is based on the Abstract Grid Workflow Language (AGWL) that allows for the creation of abstract workflows and aims to hide some of the complexities of the Grid. Askalon uses the CoG toolkit and web services to provide support for Globus. With Askalon, an abstract workflow is composed and then compiled, mapping to concrete services for execution. The GridARM console provides a visual, interactive front-end for graphically producing AGWL workflows.

GridAnt

GridAnt is an extension of Apache’s Ant tool [68] and provides a simple framework for orchestrating tasks based on XML specifications [69]. GridAnt accesses Globus through the CoG toolkit and uses the Apache Ant engine for execution. Grid Ant provides a vocabulary to specify the operations to perform. There is no visual interface for building workflows, however, a monitoring interface is available to track the progress of a workflow’s execution.

2.3.4 Workflow Comparison

Table 2.1 shows the common features between the aforementioned workflow systems. [53][72][120][121][33] present further workflow taxonomies and comparisons used in identifying common features and defining the workflow architecture. The first column identifies whether the workflow system
Background

supports a DAG (Direct Acyclic Graph) model for data flows and the type of flow, push or pull. The next two columns, control flow and control options specify whether control flow options are available and how they fit in with the typical data flow. Workflow systems that support separate layers have a co-ordination layer. Those that support nodes have special control nodes that can be used within the data flow. Links have particular constraints independent from the data, which need to be satisfied [26].
<table>
<thead>
<tr>
<th>Workflow System</th>
<th>DAG</th>
<th>Control Flow</th>
<th>Control Options</th>
<th>Grid Coupling</th>
<th>Middleware Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>InforSense</td>
<td>Yes (pull)</td>
<td>Yes (sep. layer)</td>
<td>Conditionals Loops</td>
<td>Yes</td>
<td>Globus</td>
</tr>
<tr>
<td>Taverna/Freefluo</td>
<td>Yes (push)</td>
<td>Yes (links)</td>
<td>Conditionals</td>
<td>Yes</td>
<td>GRIA, GridSAM, ARC</td>
</tr>
<tr>
<td>Triana</td>
<td>Yes (push)</td>
<td>Yes (nodes)</td>
<td>Conditionals Loops</td>
<td>Yes</td>
<td>Globus, Condor</td>
</tr>
<tr>
<td>Kepler</td>
<td>Yes (push)</td>
<td>Yes (sep. layer)</td>
<td>Conditionals Loops</td>
<td>Yes</td>
<td>Globus</td>
</tr>
<tr>
<td>Knime</td>
<td>Yes (pull)</td>
<td>Yes (nodes)</td>
<td>Loops</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>DAGMan/Pegasus</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
<td>Yes</td>
<td>Globus, Condor</td>
</tr>
<tr>
<td>GWFE</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
<td>Yes</td>
<td>Globus</td>
</tr>
<tr>
<td>Askalon</td>
<td>Yes</td>
<td>Yes (sep. layer)</td>
<td>Conditional Loops</td>
<td>Yes</td>
<td>Globus (CoG)</td>
</tr>
<tr>
<td>GridAnt</td>
<td>Yes</td>
<td>Yes</td>
<td>Conditionals Loops</td>
<td>Yes</td>
<td>Globus (CoG)</td>
</tr>
</tbody>
</table>

Table 2.1: Workflow system comparison and overview
All the workflow systems described above support DAGs for linking services and determining the dependencies. Most systems provide control flow functions such as conditionals and looping in addition to DAGs. Support for these control operations varies from workflow system to workflow system. InforSense, for example, presents control flow as a separate layer used to coordinate data flows. KNIME has specific nodes for indicating the start and end of a loop, changing the model of execution for any nodes connected between them in order to perform loops. The Director components in Kepler are capable of conditional logic; coordinating and scheduling the execution of Actors.

DAGMan, GWFE, Askalon and GridAnt are Grid-based workflow systems, specifically designed for orchestrating Grid services. GridAnt provides a vocabulary for performing Grid operations such as copy, delete, execute and status. Askalon and GWFE use a single node per service paradigm where input and outputs are defined for each node. Pegasus (with DAGMan) has a range of nodes that specify compute nodes, input data items, items transferred between compute nodes and outputs written back to long-term storage.

Unlike the aforementioned workflow systems, Taverna/Freefluo, Triana and Kepler are not specifically designed with Grid services in mind, but have been extended to support Grid functionality. Each has a different method of accessing Grid services. For example, Triana uses a three node (stage-
execute-fetch) approach built on GAT (Grid Access Toolkit) that covers the data upload, execute and download operations. Triana handles Grid invisibly to provide seamless access with its other components. In Triana, it is also possible to adopt the “one node per service” approach. [104] documents and compares briefly both node mapping approaches in Triana. Kepler builds its Grid functionality on the GriddLeS library, which handles file transfer between local and remote services automatically and explicitly, thus allowing for the one node per service view. Wendel in [109] describes an initial attempt at modelling a stage-execute-fetch approach [2] using the InforSense workflow system (using the Groovy scripting language) and Ganymed libraries for accessing Globus services.
Table 2.2: Comparison of workflow system features

Table 2.2 shows the features provided by each workflow system. Authoring client indicates whether a particular workflow system has a graphical user
interface for constructing workflows. Workflow systems with a domain view are able to deploy workflows to higher-level view, which encapsulates the inner-workings of a workflow (e.g. publishing to a web portal). Scalability indicates whether the workflow system is extensible in order to support further functionality, usually through support of third party APIs. Grid-specific workflow tools generally lack a visual interface for constructing workflows – the exceptions are Askalon, which has a visual interface for constructing workflows, and DAGMan which can use other tools such as Pegasus for an interface), which can affect usability and possibly increase the learning curve for less technical users. Note that Grid workflows also lack domain specific toolkits, designed with non-technical domain experts in mind, and the ability to extend the workflow system to support other types of services and scripting languages. As the case studies that follow show, collaborative scenarios usually demand different forms of service (be it Grid, web or local applications), rather than a single type. With their extensible frameworks and domain specific toolkits, scientific workflow systems may be better suited for these VO based scenarios.

2.3.5 Using Workflow Systems to Address SOAs

The comparison and review of workflow systems in the previous section and [121][25][109] show that scientific workflow systems have a number of advantages over existing Grid workflow systems. The WfMC architecture and
the review of workflow systems show that scientific workflow systems contain the same core layers:

- **Enactment Engine**: or workflow server that is responsible for executing the workflow and making third party calls;

- **Authoring Client**: for visually building workflows;

- **Higher-Level Domain Client**: such as a web portal that encapsulates the inner details of a workflow, providing a more scientific-functional view;

- **Extension Methodology**: supports the addition of further nodes and provides accessibility to third party applications.

The following components constitute the workflow enactment engine:

- **Data Management**: The data management package is responsible for storing data items used in workflows. These data items include tables, flat files, structured documents, references and specialized models for specific data types such as protein and gene sequences. These data items are usually stored relative to the workflow server and represented by a user space for organization.
• **Execution Management:** Execution management is responsible for handling the invocation of the submitted workflow. The execution management package determines node invocations, polls for status updates and execution progress, and handles the transfer of information from one node to the next.

• **Administration Tools:** These tools provide the ability to configure the workflow environment for optimal performance, criteria such as memory allocation, disk space available and access control. The administration tools package is also extensible, supporting customized tools for each handler.

• **Security:** The security package defines access rights to and from the workflow server. These include accessing the workflow server from remote clients and interfaces as well as proxy settings for accessing external applications from the workflow server.

• **Node Definition and Development Kit:** The node definition and development kit component supports the building of extra nodes and associated tools to be used with the workflow system. Workflow systems generally package these extra nodes and tools in handlers, which are based on specific API sets that define their interfaces.
Background

- **Composition:** The composition package is responsible for determining nodes connections and execution. It provides rules for connecting one node to the next and determines the order, distribution and dependencies associated with execution at runtime.

- **Deployment:** Deployment allows individual workflows to be published and accessible from outside the workflow system. It determines which ports and parameters are accessible to domain-based end users.

Figure 2.12 shows a detailed view of a layered workflow architecture.

![Layered workflow architecture](image-url)

*Figure 2.12: Layered workflow architecture*
Background

The authoring and domain clients are located at the top of the architecture providing interfaces for workflow users and domain experts. At the bottom of the architecture lies the handler that bridges the gap between the workflow system and third party systems, in this case Grid middleware.

2.4 Related Research Work

Research Projects

The work in this thesis uses the InforSense workflow system across a range of projects. A number of case studies from each of the projects aid the gathering of requirements for defining a Grid-workflow framework. These case studies formed the basis for the framework design, implementations and tests. The projects are as follows:

- **SIMDAT**: The SIMDAT (SIMulating DATa grids) [95][50] Project focused on enabling VOs in a secure environment by bringing together key technologies, such as Grid infrastructures, workflow systems, ontologies and knowledge services. These technologies addressed scenarios defined by partners in the Automotive, Meteorological, Aerospace and Pharmaceutical domains.

- **BRIDGE**: The Bridge (Bilateral Research and Industrial Development enhancing and Integrating Grid Enabled technologies) [14] Project was an extension of the SIMDAT Project, focusing on collaboration
Background

with partners of the CNGrid project in the People’s Republic of China. The aim was to bring the technologies extended in each of the project together to address cross-continent VO scenarios.

- **ARGUGRID:** The ARGUGRID (ARGUmentation as a foundation for the semantic GRID) [4][30] Project investigated the use of autonomous agents to perform service selection, negotiation and composition on behalf of the user. ARGUGRID required the use of an architecture, which extended the architectures used in the other projects, to address Earth Observation scenarios.

**Discovery Net Research**

Research advances in the Discovery Net project, carried out by the author’s colleagues/former colleagues, form the basis for the work conducted in this thesis. Wendel in [109][110] describes an architecture for eScience-based workflow systems and touches upon initial attempts at accessing Grid functionality from workflows, but at a more basic, higher level than the work conducted here. Giannadakis [52] focuses on frameworks for workflow client-based extensions, and on workflow abstraction through lambda lifting, a platform for the Grid workflow abstraction implementations described in this thesis. Curcin [25][26][28] investigates workflow patterns and matching based on types. The work conducted in this thesis builds on some of these type matching principles and patterns and applies them to Grid-based workflow designs. Osmond [88] focuses on portal deployment and
extensions for the InforSense system, which were required for user role-based abstractions and implementations (i.e. portal based administrative tools and workflow publishing). The lessons in Rowe [93] regarding the designs for building workflow-based scientific applications are applied here in each of the EU projects. Syed [101] presents a structural analysis and definition of the underlying XML based representation of InforSense workflow, necessary for the workflow abstraction and representation work conducted here.

2.5 Motivation

Grid workflows are designed to bring Grid and workflow systems together in order to address SOAs. However, as described in this chapter, they are inadequate for addressing the usability needs of the domain expert, therefore scientific workflows, designed for particular domains, are preferred. While some scientific workflow systems support Grid service access, the work carried out with respect to these systems is typically not generalized, and specific to certain workflow and Grid implementations.

In terms of addressing the Grid-Workflow gap, a number of general methodologies are currently available. For example, the CoG toolkit aims to address to fill in the gap between the two technologies by supplying an API for workflow systems to connect to multiple Grid servers. However, the CoG
Background

toolkit design is from a Service Provider perspective and does not indicate how workflows nodes should be mapped to the CoG API. The work carried out in [2] identifies key workflow components to access Grid services, but does not indicate how these components are used or identify the flow of information between them.

Based on the review of these current methodologies this thesis proposes a more general and flexible framework, which takes into account various forms of node mappings and interactions between architectural components across all Grid and workflow layers.

2.6 Building Workflows

The workflows shown throughout this thesis are developed using the InforSense workflow system. Figure XX shows the Workflow Builder (user interface). Users can drag nodes from the component tree on the bottom left to the workspace in the middle. The node editor at the bottom of the screen allows displays the parameters for the selected node. Users can edit or configure these parameters before workflow execution. Further nodes can be dragged to the workspace and connections between can be made if an output port meets the metadata requirements of the input for the next node (e.g. if the input is of the same type or contains the required columns). A user can execute a complete workflow, invoking all connected nodes from left to
Background

right (direction of the connections). Each node execution produces an output that the workflow system caches and has to be explicitly saved.

![Workflow Builder Layout](image)

**Figure 2.13: Layout of InforSense Workflow Builder for constructing workflows**

The nodes developed as part of the work conducted in this thesis are built in this way and follow the same pattern of execution.
2.7 Summary

This chapter presents an overview of the SOA paradigm and the concept of Services, Service Requestors and Service Providers. The identified Grid and Web service standards are an integral part of the SOA model, particularly from the Service Provider perspective. Workflow systems satisfy the Service Requestor perspective, but lack a methodology for providing extensions to bridge the two halves and build a complete framework that addresses all the layers of the SOA model. With existing Grid flows and standards failing to provide a reusable flexible solution that accommodates proprietary workflow tools, this thesis presents a framework for Grid enabling current workflow systems. Figure 2.14 shows a complete SOA based layered architecture that encompasses workflow systems and Grid middleware. Note that this architecture diagram shows only a single workflow client connecting to a single workflow server and only accessing one Grid system for simplicity. As shown in the case studies presented in this thesis, a single workflow server can connect to multiple Grid systems, and accessed by multiple workflow clients.
Figure 2.14: Combined workflow and Grid layered architecture in accordance with the SOA paradigm

The next chapter looks at addressing the Grid handler definition, investigating the layers and components that make up the Grid handler and the interactions between them.
Chapter 3
Mapping Grid Functionality to Existing Scientific Workflow Systems

This chapter investigates the design of a basic Grid handler for accessing Grid functionality via workflow systems. A key challenge arising when designing a Grid based workflow system is deciding what constitutes an atomic step in a workflow. Ideally, to enhance usability, each node should correspond to one remote service. However, in practice this may not be achievable. This chapter investigate the factors affecting the choice of node mappings and investigate their effects on workflow system architecture.

The framework designed in this thesis for addressing the integration between workflow and Grid systems is primarily built around identifying the Grid functionality to be supported by the workflow system, mapping functionality to workflow nodes while maintaining usability, identifying supporting interfaces that facilitate the node definitions and identifying the interactions between the components across all layers. Three potential forms of nodes mappings are presented with their advantages and disadvantages in terms of usability, functionality supported and performance identified. The framework
is tested and evaluated using collaborative case studies defined in the SIMDAT project, which are spread across a range of domains.

3.1 Supporting Grid Functionality from Workflow Systems

Given the combined workflow/Grid layered architecture and user role models defined in Chapter 2, workflow systems are not required to support all available Grid functionality directly. Grid clients can be used in tandem with workflow systems at the administrative level, i.e. to provide low level configurations to support workflow authoring. Administrators are required to determine security and access settings based on user roles, as Grid service calls occur from the workflow enact engine irrespective of the location of the workflow clients. Domain experts are responsible for simply executing the services and therefore are not required to produce workflows. An analysis of the general steps for running a Grid process can help to determine how and what functionality is accessible to the workflow author role.

The following steps described the execution process for a Grid service and must be mapped to workflow nodes, allowing workflow authors to co-ordinate Grid services. The first step in determining any form of node mapping is to identify the Grid server with which to establish contact, in order to search for and bind to hosted and deployed Grid services. After determining the Grid
server, trust and security need to be negotiated to validate the end user’s authorization rights with respect to the services made available by the service provider. Upon establishing trust, the next step is to identify and select a service to access. After service selection, the inputs, outputs and parameters for each service require configuring in order to determine the type of, and transfer processes for, data to and from the service. This configuration process has significant importance during workflow authoring, determining which nodes are required to pass data to and process data from a set of Grid service execution nodes. Utilizing service metadata provided by some Grid systems, can help aid this process. Any data required by a Grid service must be made accessible to it, which can be achieved in two ways: moving the service to either the data or the data to the service, the latter most commonly used for remote services, as applications are usually tied to the hardware and software where they are installed. After identifying a service and its ports and parameters, it is necessary to determine the resources required to execute the service and billing details. Resource allocations include variables such as the data storage space, number of CPUs, etc. The transferring of any input data (required by the Grid service) to the server occurs after the allocation process. The service (job) is then instantiated. Submission of arguments along with references to the uploaded data for execution can occur after the instantiation of the required service. Job execution logs the status of the job and progress of the underlying application. When the job has finished executing successfully, the Grid
server returns references to the output data items to client applications. These client applications can then download the data to a local machine.

Based on the aforementioned standard Grid service execution process, it is possible to identify the main Grid operations required from a workflow author’s perspective, namely the allocation and management of resources, data upload to the Grid server, execution of a service and retrieving results from a Grid server. Any additional Grid functionality, such as billing and access rights can be addressed from an administrative perspective using specially constructed tools within a workflow system or available Grid clients.

3.2 Grid Handler Definitions and Node Mappings

The Grid handler plays a vital role in the context of a workflow/Grid architecture and is designed around two APIs. The Handler API provided by the workflow system defines the methods required to produce and define nodes. The Grid API, which provides access to the Grid services, defines interfaces that the workflow must utilize in order to perform the operations outlined in the previous section. Like the rest of the architecture, the Grid handler definition is a set of layered components. Figure 3.1 shows this layered architecture for a Grid handler.
<table>
<thead>
<tr>
<th>Client</th>
<th>Portal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>Interface</td>
</tr>
</tbody>
</table>

### Workflow Server

<table>
<thead>
<tr>
<th>Data Management</th>
<th>Execution Management</th>
<th>Administration Tools</th>
<th>Security</th>
<th>Node Definition &amp; Development Kit</th>
<th>Composition</th>
<th>Deployment</th>
</tr>
</thead>
</table>

### Handler API

**Grid Handler**

- **Node Definition**
- **Workflow Based Support Interfaces**
- **Grid Based Support Interfaces**

### Grid API

**Grid Interfaces**

<table>
<thead>
<tr>
<th>Resource Allocation</th>
<th>Data Service</th>
<th>Job Service</th>
<th>Application Metadata</th>
<th>Access Management</th>
</tr>
</thead>
</table>

**Wrappers**

- **Data Storage**

- **Application**

- **Computational Resources**

---

Figure 3.1: Layered Grid-workflow architecture with Grid handler layer definition
The top layer of the Grid Handler provides the node definitions. These node definitions perform modularised Grid functions that can be co-ordinated to perform service executions and determine connectivity with other workflow nodes. The middle layer houses any supporting classes or objects required by the node definitions. The design of the middle layer is from an object-oriented perspective, in that abstraction of common functionality promotes reuse and avoids duplication. The bottom layer represents the components (such as libraries, configuration files and objects) provided by the Grid vendor, allowing third party applications to access the Grid.

The Grid handler definition is based on the assumption that workflow handlers are structured self-contained entities, such as those supported by InforSense, Taverna, Triana and Kepler. For workflow systems such as KNIME where only the node definitions are self-contained, the same structure can apply, however the Grid support interfaces must be added to the workflow server directly.

One of the main challenges associated with determining node mappings is how to maintain a level of node usability while supporting the required Grid functionality. Supporting a large subset of Grid functionality from the node mappings, which would involve revealing low-level technical details to the workflow author, allows for a wider range of workflow solutions to be built. In doing so, the node mappings can become complicated and difficult to use, leading to an increase in workflow authoring time and requiring workflow
authors with a greater technical skill set. Adopting a lower level form of node mappings would take longer to develop and troubleshoot, and provide more points of failure. Reducing the functionality supported by node mappings provides simpler interfaces for use by a wider, less technical range of workflow authors and end users, quicker workflow turnaround, quicker development time and less points of failure. The disadvantages associated with simpler mappings include less flexibility and infeasibility for complex solutions, due to the lack of supported functionality and configuration of low-level Grid parameters.

Node mappings can either be generic, i.e. applicable for many services, or tied (in some cases hard-coded) to particular services. Like web service and local node mappings, the Grid mapping design presented here is based around a generic approach rather than specific, tied mappings, which provides support for a wider range of services, promotes node reuse and is suited for handling service implementations that change or evolve over time.

This chapter describes three forms of node mappings. Figure 3.2 shows how they are used to execute a single Grid service.
The Single node mapping involves a single node to encapsulate all required Grid functionality. The R-DTE (Resource allocation, Data Transfer and service Execution) mapping separates the Grid functionality into nodes. The DTE-M (Data Transfer, service Execution and Metadata handling) mapping also uses a node separation but removes the resource allocation process from the workflow authoring level and introduces metadata handling for node configuration.
3.3 Single Node Mapping

The simplest method for mapping Grid functionality to nodes follows the one node per service approach generally adopted for local and web services. A single node-based mapping requires parameters for specifying security and authorization settings (i.e. locating key stores, passwords and alias names), a way of defining and/or providing resource allocation variables, and parameters for specifying input arguments, Grid server and service endpoint. Workflows rely on metadata to streamline the workflow authorization process, thus for a single node approach, the input and outputs are required to handle flat file path references rather than passing files directly during authoring time. Using file references also allows the Grid node to interact seamlessly with other components that handle file paths such as command line bases nodes and nodes that import/export files to and from locations accessible to the workflow system.

3.3.1 Single Node Grid Handler

Figure 3.3 below shows the expanded Grid Handler layer with the components required to implement the single node mapping.
3.3.2 Single Node Parameters

The parameters for the Single Node to support the described functionality fall under the following categories:
Security

Most workflow systems are designed around a client/server architecture, thus the workflow server is responsible for performing the service execution. Configuring the security information directly from the node is not feasible given that some workflow authors may not have the required administrative rights to configure these settings. The handling of Grid security settings occurs outside the node interface and through a configuration file or object accessible only at the administrative level. The security settings supported from a single node mapping can include the handling of certificates, keystores and their associated passwords, as well proxy settings.

Resource Allocation

Having abstracted the security settings to an administrative level the first set of node parameters associated with the single Grid node relate to resource allocation. The method for specifying the resource allocation metrics is dependent on the Grid middleware and API used. Typically, the single Grid node can support either separate parameters for each metric or a single parameter/input that allows for the specification of metrics within a flat file. The resource allocation process itself can occur at either authoring time or runtime. Performing an allocation at runtime must generally be an automated process, unless the workflow system used provides support for runtime. When using this form of runtime allocation, the node must resolve exceptions where allocation requirements remain unsatisfied by either terminating the execution of the workflow or accepting automatically the next best resolution.
Utilizing an automated runtime selection mechanism requires the use of extra metrics to determine the criteria for selecting an allocation, i.e. the cheapest allocation or the quickest allocation. An authoring time allocation process allows the workflow author to interact with the Grid server in order to refine and select allocations before executing the workflow.

**Job Settings**

A job instance refers to the invocation of a single Grid service and a range of parameters associated with the single Grid node are required to perform this invocation. First, the single Grid node must provide a parameter for specifying the job service endpoint, which identifies the Grid server to connect to. Second, another parameter is required to allow the workflow author to specify the service URI. An identification name can usually be applied to an instance of a job service. The single node should therefore provide a way of specifying or automating a reference name for a job instance. Some Grid middleware systems also allow the specification of additional job arguments to from a client governing the execution of the service. Like the resource allocation process, the single node definition interface can provide additional parameters for specifying these arguments.

**Inputs and Outputs**

In order to represent the flow of data to and from a Grid service, individual ports are required for each input/output. When using a general approach like the single node mapping, specifying and configuring these ports is a dynamic
process, i.e. the node cannot come fully predetermined before identification of the service to execute. The single node approach must therefore provide a way of specifying the number of input/output ports and allow configuration of each port separately. For each input port generated, a parameter for specifying a reference name is required in order to track and identify a data item copied to the Grid server. To determine the location where the outputs from a Grid service execution are copied, a path parameter can be provided with each output port generated. Along with the standard input and output ports related to data transfer, the addition of further ports supports the transfer of additional (or optional) service arguments or configuration files, and to return application logs at the end of each service execution.

**Node Process Design**

The pseudo code example in Figure 3.4 illustrates a potential node process method for invoking a service based on the single node approach. Note that in this particular example obtaining the resource allocation occurs automatically at runtime.
Mapping Grid Functionality to Existing Scientific Workflow Systems

```
GET resourceAllocationParams from parameters OR READ from file
GET parameter with dataServiceEndpoint RETURNING dataServiceURL
GET parameter with jobServiceEndpoint RETURNING jobServiceURL
GET parameter with numberOfInputs RETURNING noOfInputs
GET parameter with numberOfOutputs RETURNING noOfOutputs

GET new instance of stateRepository RETURNING repository

BEGIN
    CALL repository.getCheapestAllocation with resourceAllocationParams RETURNING resourceAlloc
    EXCEPTION
    WHEN no allocation can be made
        End node execution and request new resourceAllocationParams
    END

dataItems array with length numberOfInputs
FOR each input
    CALL repository.newDataService with dataServiceURL and resourceAlloc RETURNING dataService
    CALL dataService.newDataItem with inputReferenceName RETURNING dataItem
    GET inputFile from nodeInputs
    CALL dataItem.save with inputFile
    ADD dataItem to dataItems
ENDFOR

CALL repository.newJobService with jobServiceURL, resourceAlloc RETURNING jobService
CALL jobService.newJobInstance with jobReferenceName RETURNING jobInstance
CALL jobInstance.submitJob with serviceEndpoint, arguments and dataItems

WHILE jobInstance is running
    CALL checkJob
    GET application log
ENDWHILE

IF jobInstance is unsuccessful
    Stop node execution
    RETURN log and errors
ENDIF

FOR each output invoked
    CALL jobInstance.setOutput RETURNING dataItem
    new file with outputFilePath
    CALL file.write with dataItem.read
    ADD outputFilePath to output
ENDFOR
```

**Figure 3.4:** Pseudo code example of the Single Node’s process method for invoking a Grid service

### 3.3.3 Single Node Support Interfaces

Technically, it is possible to build directly all the Grid functionality directly into the node definition. However, extrapolating common functionality into separate helper objects or interfaces utilizes the advantages of object-oriented abstraction. These objects are usually associated with persisting
values for later reuse or recollection. With the single node mapping, these support classes are based around common functions used by both the single Grid node and the administration interfaces.

**Administration and Configuration**

The administration and configuration helper object provides a foundation for any administrative tools specifying the security settings associated with the Grid handler. The helper object is required to perform error handling and validation checks on the security setting in order to flag any incorrect or irresolvable values, e.g. malformed URLs, incomplete file paths, etc.

**Security**

The security helper object (used in tandem with the administration and configuration object) is built upon the security interfaces specified in the Grid middleware API. This helper object is required for performing the error handling and validation operations that are associated with and require contact with the Grid middleware. These operations include submitting to the Grid server security settings, such as proxy values and keystore information, to authenticate the workflow system and establish connectivity before node execution can begin. It is possible to provide the functionality covered in both these helper objects in a single object, however by providing the distinction, the administration and configuration validation checks can be used with other forms of settings, i.e. those associated with web services or other applications.
3.3.4 Single Node Interactions

In order to understand how the single node mapping components work together to perform a Grid service execution, it is necessary to identify the interactions that take place across the different architecture layers. The following interaction diagrams describe the communication between the various components during administration, workflow authoring and workflow execution.

Administration and Configuration

Figure 3.5 shows the interactions that take place at the administrative level in setting up the workflow environment. These integration diagrams extend the UML communication diagram standard. Communication diagrams are suitable for describing the interactions presented here as they map to the layers and components of the architecture.
Figure 3.5: Single Node interactions for Administration and Configuration operations required before workflow authoring and execution

An administrative interface can be a client/portal-based application provided as part of the workflow system or an external third party application built around the workflow API. This interface allows for security, proxy details and the transfer of any required files by a user with administrative rights. The Administration and Configuration helper object submits and persists these values, and performs basic error checking and validation. If the administrative tools return no errors during the validation process, the values entered pass to the security object, which is then responsible for establishing connectivity with the Grid middleware specified. Upon successful contact, the
values must then be stored and persisted for use in the service execution process.

**Execution Interactions**

Figure 3.8 identifies the interactions performed across the architecture upon execution of a workflow consisting of a single Grid service. This particular interaction diagram represents an automatic resource allocation resolved at runtime.

![Component interactions for executing a Grid service using the Single Node approach](image-url)
From the workflow client, an author parameterizes a single Grid node in terms of resource allocation, inputs/outputs and job-based arguments identifying the Grid server to use and the service to invoke. This Grid node connects to other workflow nodes that handle file references to and from the service. Executing the workflow persists the security settings entered at the administrative level for establishing connectivity with the Grid server. The Grid Handler submits the allocation parameters to the Grid server via the Resource Allocation Grid interface. The Grid node retrieves and stores the best allocation returned for use with this execution. The node reads file references from any input ports and the file is copied to and managed by the remote Grid server using the Grid data service. The Grid server returns a reference to each data item uploaded from the workflow for use with the job service. The workflow submits service endpoints, data references and arguments to the Grid job service. The job service invokes the application using the data specified and the polling process begins to determine the status of the job. Any log returned by the job service or application are systematically updated as the job progresses. Upon job completion, the Grid data service manages the output data produced, returning references for each output to the workflow system. The Grid server transfers each job output to a location specified in the node parameters corresponding to a file system accessible via the workflow system. When the node downloads all the outputs, the execution finishes.
3.3.5 Single Node Implementation

This section describes the application of the single node mapping design to the InforSense workflow system for implementing a plug-in (handler) for accessing GRIA middleware. At the administrative level, a customized GRIA plug-in configuration interface for the InforSense portal uses a set of web pages using JSP (JavaServer Pages) technology bundled with the plug-in and is only accessible by users with administrative rights. The portal page includes fields to specify the path to a keystore, along with its associated passwords and aliases, required by GRIA for establishing access. The configuration interface provides fields to select an existing or create a new client.state file (a structured XML document that stores information of job and data instances). It is also possible to specify HTTP/HTTPS proxy settings portal if required. Validating these fields using the administration and configuration support component ensures that no compulsory information is missing, entered incorrectly or irresolvable. Submitting the security settings from the portal invokes the security support component, which generates underlying configuration files within a folder contained in the GRIA plug-in to persist the information. Binding these files within the plug-in allow the plug-in to be transferred to different machines without losing current settings. The administrative tools send any entered proxy settings to the GRIA server using the security helper for validation. Figure 3.7 shows the portal configuration page for the GRIA handler.
Figure 3.7: InforSense portal based configuration page for the GRIA handler

Node definitions in an InforSense plug-in are Java based, thus the single Grid node for accessing GRIA services uses the Java API provided by GRIA. The API supports a Java binding for service operations, helper classes to encapsulate complex tasks and Java Swing interfaces for browsing allocations and monitoring jobs from a client (such as a Grid workflow node). These operations generate the appropriate SOAP based messages using SwA. The plug-in packages GRIA client libraries, which provide these APIs and facilities, for easy porting to different InforSense installations as described in the bottom layer of the Grid handler design.
The single node consists of parameters that relate to resource allocation, data upload, job execution and result download. The workflow author provides resource allocation parameters via an XML file. Resource allocation is either a manual or an automatic process. Automatic allocation happens at runtime, where the node selects the best allocation based on the criteria specified in the configuration file, e.g. the cheapest allocation. Manual allocation happens during workflow authoring. From the node interface, selecting a manual resource allocation process will generate a parameter that launches a Java Swing GUI provided by GRIA API. When the GUI is launched from the node, the node creates a DOM object from the resource allocation XML file. The node then parses the values and invokes an API call which generates and submits a SOAP message for locating available resources. A list of available allocations populates the GUI, indicating which resource allocations requirement are or met or unmet. The user can then select an allocation, invoking another SOAP call which returns a reference that is persisted throughout the node to track the status of the resource allocations.

Dynamic parameters handle input and output port definitions. Two parameters allow the workflow author to specify the number of input and output ports to include relative to the service implementation. Each input, generates a port that accepts a file reference object, which points to a file on file system accessible from the workflow server. At runtime, the node generates a Java File object for each file referenced. These files are added
as attachments to the SOAP message for the file upload operation. A parameter is also generated for specifying the data service endpoint and a parameter for specifying a reference name used to identify the data item once it is copied to the Grid server. Likewise, for each output the node generates a port (which produces a file reference) along with parameters for specifying the download location.

The GRIA node provides job parameters for specifying a job service endpoint, service URI, job reference name and job allocation (stating how much of the resource allocation is expected to be used by the job). Note that these job and input/output parameters include error handling mechanisms, some of which are part of the InforSense workflow system and some of which are coded specifically, e.g. checks to ensure the service endpoints and URIs are well formed. Figure 3.8 shows an example of a simple workflow for executing a Grid-based image manipulation service. The single GRIA node parameters associated with performing resource allocations are shown in the panel below the workflow. The first node in the workflow is a standard InforSense node for locating a file. The output is a file reference passed to a GRIA node which submits the file to the GRIA server, executes the service, downloads the result and returns a file path reference. The final node is a standard GRIA node for displaying the returned file.
Execution of the GRIA node (based on automatic resource allocation) follows the same pattern as described in the interaction diagram Figure 3.6. During the polling process, InforSense retrieves the application logs to identify the current job and application status. Successful execution copies the output files as expected. However, cancelling the node execution must invoke the GRIA wrapper script that halts the service execution to synchronize the
cancellation of the node and service. This operation prevents GRIA jobs from running independently of GRIA node instances.

### 3.3.6 SIMDAT Case Studies: Grid-Enabled WEKA and Crash Simulation

The InforSense/GRIA implementation is applied to collaborative scenarios as defined in the SIMDAT project [10][50] to test the validity and evaluate the single node mapping-based framework. The first scenario focuses on providing a Grid-enabled version of WEKA (Waikato Environment for Knowledge Analysis) data mining tools, which are a collection of machine learning algorithms. This knowledge service scenario is ideal as an initial test for the Grid-workflow framework since it focuses on seamlessly integrating a single Grid service with other workflow nodes and services, such as web services. The second scenario focuses on an automotive collaboration for performing crash simulations within a VO setting. Whereas the knowledge services scenario focuses on a single Grid service, the automotive scenario focuses on chaining multiple Grid services together.

**Case Study Environments**

For the Grid-Enabled WEKA scenario, Grid versions of WEKA tools are deployed using GRIA v4 with each tool distributed in a Condor pool at a research institute in Germany. A standalone installation of InforSense v2 (workflow client and server running on the same machine) is deployed in the
UK. This InforSense installation includes a GRIA v4 plug-in based on the single mapping Grid handler design to support the creation and invocation of Grid workflows.

The automotive Crash Simulation scenario is a collaborative VO consisting of an OEM (Original Equipment Manufacturer), a supplier for managing development and design of products and a trusted third party for providing the host environment. The supplier (located in France) has an installation of InforSense v2 with a GRIA v4 plug-in installed and a GRIA v4 client. The crash simulation tools are deployed using GRIA v4 at the trusted third party in Germany on a four process Linux 64 bit machine.

**Grid-Enabled WEKA Scenario**

The Knowledge Services domain (within the context of SIMDAT) aims to provide a Grid-based solution for accessing their customized WEKA algorithms. These algorithms are available as configurable and downloadable libraries, accessible to authorized users. These libraries form the foundation for building further high-level data mining applications. Due to authorization required to access the libraries, the download operation is better suited to Grid deployment rather than to standard Web services. To determine which library to download, the Knowledge Services partners provide a standard web service that takes ontological terms and returns a matching library. Web services are more appropriate for this particular operation, due to its handling of only simple object types. The overall
objective of this scenario is to perform an initial test of the single node implementation as well as evaluate the design of the node in terms of integration with other workflow nodes, and compatibility with other remote service technologies.

Figure 3.9: Grid-Enabled WEKA workflow demonstrating seamless connectivity between Grid node and other workflow nodes

The workflow designed to address the Grid-enable WEKA scenario consists of two main nodes, with a number of pre and post processing nodes as shown in Figure 3.9. The first main node is a web service node that takes ontological terms as string values and returns values to build a URI to a
downloadable WEKA library. The second main node is a GRIA node, configured to take a single string (based on a parsing and concatenation of the outputs from the web service node) as an argument for the GRIA service. No file-based inputs are associated with the GRIA node and it has a single file reference output that points to the configured and downloaded WEKA library.

**Crash Simulation Scenario**

Computer model automobile crash simulations aim to reduce the cost and time of prototypic real life testing. The Automotive Crash Simulation scenario involves a CAD model of a car part that combines with a CAD model of the surrounding parts provided by the car manufacturer. The simulations crash this combined CAD model with various customizable load cases. Varying the size, weight and material of these load cases, as well as the speed and angle of crashes, provides a wider range of results. The part designer accesses these crash results to evaluate and analyze the performance, strengths and weaknesses of the designed part. Performing crash simulations requires the use of HPC intensive jobs and large, complex data files and is thus suited to Grid computing. The security features provided by Grid middleware are also essential for such a scenario, as proprietary CAD models from different organizations require strict access rights and results should be available only to the part designer.
The workflow to address the crash simulation scenario must support the execution of two GRIA services as shown in Figure 3.10. The first service combines the CAD models provided by the part designer and car manufacturer to ensure that the models fit, or to flag any potential errors or problems before the execution of expensive simulations. The second service takes this combined model along with a configuration file specifying the load cases, and performs the actual crash simulation. The nodes in between represent localized pre- and post-processing operations required to transform or produce required data.

**Evaluation of Scenarios**

Both scenarios successfully demonstrate the applicability of the single node mapping design for accessing Grid services from a workflow system within domain specific scenarios. The Knowledge Services scenario shows that the Grid node definition allows for the seamless integration of Grid functionality
with other workflow nodes and tools. The automotive scenario demonstrates how composing Grid nodes can help form higher-level applications. The latter scenario, however, also highlights a number of problems associated with the single node mapping. First, by encapsulating all the steps in the Grid service execution process within a single node, negates the stateful nature of Grid services leading to erroneous data upload and download processes between Grid nodes, which can incur extra resource allocation costs. For example, before the crash simulation tool is instantiated, the workflow downloads and uploads the combined car model produced by the combination tool to the same GRIA server, even though the passing of a data reference would suffice. Discussions on the effect these extra processes have on workflow performance appear later in this chapter. Second, changing parameters or arguments in the single node clears the node cache allowing for re-execution. Clearing the cache also clears any data references, leading to data duplication as the workflow copies files over to the Grid server, regardless of the fact that the data from the initial copy is still available there. Service failure during mid-workflow execution would also require re-execution, leading to the same problem. With the current resource allocation process, defining allocations can become time-consuming as the number of Grid nodes within a workflow increases. Both scenarios show that designing the framework around the user role distinctions eases the workflow configuration, authoring and execution process by limiting the functionality available to each role. The scenarios indicate that moving the handling of data and job service endpoints to the administrative level improves security.
Doing so limits the Grid servers accessible to workflow authors, thus preventing the submission of trusted data to unauthorized servers.

### 3.4 R-DTE Mapping

The R-DTE (Resource allocation, Data Transfer and Execute job) mapping addresses some of the shortcomings of the single node mapping, garnered from the evaluation of the case studies. The R-DTE mapping splits and distributes the Grid functionality across multiple nodes, i.e. single nodes for resource allocation, data upload, job execution and data download, respectively, and the introduction of models or objects for referencing data on the remote Grid server and resource allocations.

#### 3.4.1 R-DTE Grid Handler

Figure 3.11 shows an R-DTE-based Grid Handler with refined node definitions and support interfaces required to address the identified single node mapping issues.
The node definition layer extends the single node handler to support individual nodes for each of the key operations, with additional objects/models to persist and transfer the resource allocation and data references between nodes. To support these extra nodes, helper interfaces abstract common functionality previously handled in the single Grid node definition.

### 3.4.2 R-DTE Node Definitions

With the exception of Administration and Security, the node definitions for the R-DTE mapping need to be supported the following categories.

**Administration and Security**

Similar to the single node mapping, the R-DTE mapping requires
administration interfaces to specify the security settings used for establishing contact with a remote Grid server. The administrator's responsibilities, with respect to the single node interface, must now include the specification of data and job service endpoints accessible to workflow authors via the node interfaces.

**Resource Allocation**

The R-DTE mapping conducts the resource allocation process through a single node, which forms the starting point of any Grid workflow or subworkflow. The parameters for this node mirror the allocation parameters defined in the single node approach. As before, the node can retrieve allocations during workflow authoring or automatically at runtime. Separating the resource allocation process from the rest of the Grid functionality requires the propagation of each allocation through the workflow. Allocation propagation is achievable by using a model or specialized object to pass the resource allocation to the other nodes. Note that the resource allocation node can support an optional input that does not propagate any data but acts as a synchronization port, allowing resource allocation node to start in the middle of a workflow.

**Upload Data**

The upload data node transfers a single file from a file system accessible by the workflow server to the Grid server. Using a one node per data item approach allows for state utilization and thus promotes data reuse. An
upload data node requires a file reference object and a resource allocation model/object, given the separation of the resource allocation operations. Data service endpoints specified by an administrator must be visible from the node interface to allow the workflow author to determine which Grid server will receive the data. A node can support this functionality by providing a choice or list parameter that reads from an object or file containing the values entered by the administrator. Similar to the resource allocation node, a dedicated model/object handles the propagation of the data reference returned by the upload data node.

**Job Execution**

The job execution node explicitly handles the invocation of a service. Like the single node approach, dynamic ports are required to mirror the service inputs and outputs from the workflow. The node supports data reference model/object types, rather than file references, to align with the other nodes defined in the R-DTE approach. The passing of these data reference model/objects to and from the job service node and indeed the other nodes removes the unnecessary download and upload operations that were problematic in the single node approach. This node definition requires additional input and output ports to support the propagation of resource allocations. The parameters for such a node can follow the job service parameters for the single node approach. However, like the upload data node, changes to support administrator-based endpoint specification are required.
Data Download

The data reference object transfers and persists references to data items located on a Grid server. A persistable object supports data reuse and provides a fail over mechanism if the workflow or Grid server fail mid-execution. These benefits are not restricted to the single workflow enclosing the data reference object. Most scientific workflow systems provide a way of saving customized objects, thus it is possible to use a data reference saved by one workflow in another workflow.

Data Reference Object

The Data Reference object transfers and persists references to data items located on a Grid server. Using a persistable object supports data reuse and provides a fail over mechanism if the workflow or Grid server fail mid-execution. These benefits are not restricted to the limits of a single workflow. Most scientific workflow systems provide a way of saving customized objects, thus it is possible to use a data reference saved by one workflow from another workflow.

Resource Allocation Object

The resource allocation object propagates resource allocation through a workflow or sub-workflow. This product of the resource allocation node also propagates across the other nodes in the R-DTE mapping.
Nodes Processes Design

The pseudo code examples in Figure 3.12, Figure 3.13, Figure 3.14 and Figure 3.15 exemplify possible process method designs for the resource allocation (manual process), upload, execute and download nodes respectively.

```
RESOURCE ALLOCATION
GET resourceAllocationParams from parameters OR READ from file
Get new instance of stateRepository RETURNING repository
BEGIN
    CALL repository.getAllocations with resourceAllocationParams
    RETURNING resourceAllocArray
EXCEPTION
    WHEN no allocation can be made
    End node execution and request new resourceAllocationParams
END
DISPLAY allocations in resourceAllocArray
GET selectedAllocation
CALL repository.getAllocation with selectedAllocation
RETURNING resourceAlloc
CALL resourceAllocModel.newInstance with resourceAlloc
RETURN resourceAllocModel
```

Figure 3.12: Pseudo code for Resource Allocation node design, producing a resource allocation model
<table>
<thead>
<tr>
<th>UPLOAD DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET parameter with dataServiceEndpoint RETURNING dataServiceURL</td>
</tr>
<tr>
<td>Get new instance of stateRepository RETURNING repository</td>
</tr>
<tr>
<td>Get resourceAllocModel from nodeInputs RETURNING resourceAlloc</td>
</tr>
<tr>
<td>GET inputFile from nodeInputs</td>
</tr>
<tr>
<td>CALL dataServiceMgmt with dataServiceURL and resourceAlloc RETURNING dataService</td>
</tr>
<tr>
<td>CALL dataService.newDataItem with inputReferenceName RETURNING dataItem</td>
</tr>
<tr>
<td>CALL dataItem.save with inputFile</td>
</tr>
<tr>
<td>CALL dataRefModel.newInstance with dataItem.getRef</td>
</tr>
<tr>
<td>CALL resourceAllocModel.update</td>
</tr>
</tbody>
</table>

RETURN dataRefModel
RETURN resourceAllocModel

**Figure 3.13:** Pseudo code design for Upload data node that takes a resource allocation model and produces resource allocation and data reference models
EXECUTE JOB

GET parameter with jobServiceEndpoint RETURNING jobServiceURL
Get resourceAllocModel from nodeInputs RETURNING resourceAlloc

dataItems array with length numberOfInputs
FOR each input
  CALL dataServiceMgmt.getDataItem with dataRefModel.getRef
  RETURNING dataItem
  Add dataItem to dataItems
ENDFOR

CALL jobServiceMgmt.newJobService with jobServiceURL, resourceAlloc
RETURNING jobService
CALL jobService.newJobInstance with jobReferenceName
RETURNING jobInstance
CALL jobInstance.submitJob with serviceEndpoint, arguments and dataItems

WHILE jobInstance is running
  CALL checkJob
  GET application log
ENDWHILE

IF jobInstance is unsuccessful
  Stop node execution
  RETURN log and errors
ENDIF

For each output
  CALL jobInstance.getOutput RETURNING dataItem
  CALL dataRefModel.newInstance with dataItem.getRef
  SET nodeOutput to dataRefModel
ENDFOR

CALL resourceAllocModel.update

RETURN selected dataRefModel
RETURN resourceAllocModel

Figure 3.14: Pseudo code for Execute job node design that takes and produces data and resource allocation reference models
### 3.4.3 R-DTE Support Interfaces

The R-DTE Grid handler utilizes the following Support interfaces, designed around common node functionality, namely resource allocation, data management and job management. Note that these interfaces extend the helper interfaces defined for the single node mapping.

#### Administration and Configuration

Extensions to the administration and configuration interfaces are required to support the entry of Data and Job Service endpoint parameters from an administrative level. As with the other security settings, validation performed on the endpoints ensures they are well-formed.

#### Data Management

A data management helper interface handles the data service-related
operations associated with the administration interfaces and node definitions through contact with the Grid data services. In order to establish contact with Grid servers, the data management object must work in tandem with the security objects.

**Job Management**

Similar to the data management object, the job management object works with the security and administration interfaces to provide validation for the job service endpoints. In the R-DTE approach, this object houses only the methods required for the administrative interfaces and job executions nodes to interact with the Grid job services.

**Resource Allocation Management**

The resource allocation node and the resource allocation model use the resource allocation management interface primarily for establishing allocations with the remote Grid server. Other nodes support this interface in order to persist the allocation across a workflow.

**3.4.4 R-DTE Interactions**

The following interaction diagrams describe the communications that occur within the context of the R-DTE mapping. Note that due to the modularization of the Grid steps, and support for further Grid functionality such as state
Mapping Grid Functionality to Existing Scientific Workflow Systems

management, the interactions for the R-DTE approach are far more complex than the interactions associated with the single node approach.

Administration and Configuration

The interactions shown in Figure 3.16 describe the administrative process for specifying and validating the job/data service endpoints before workflow authoring takes place.

![Figure 3.16: Component Interactions for Administration and Configuration](image)

Resource Allocation

Figure 3.17 shows the interactions involved in retrieving a resource
allocation. This design of this particular interaction supports both runtime and authoring time allocation selection.

The workflow author specifies the allocation parameters in the node interface. The node sends these allocations parameters to the management object, which is responsible for parsing the information, and building any allocation objects required by the Grid server. Using the security settings, the management objects establish a connection with the resource allocation interface associated with the Grid middleware. A selected allocation (whether manually or automatically) returns to the management object to build an
allocation object to be passed along the workflow. The node returns an instance of a resource allocation object with the returned allocation.

**Upload Data**

Figure 3.18 shows the interactions associated with uploading data to a Grid server using the R-DTE mapping.

![Diagram of interactions for executing a R-DTE Upload Data node](image)

Figure 3.18: Interactions for executing a R-DTE Upload Data node

A resource allocation object and file reference passes along to an instance of the upload data node. The Upload data node then submits the details to the Data management interface, which then contacts the Grid data service using
the configured security settings. If a connection with the Grid server is established, the Data Service creates a new data instance and copies the file referenced in the workflow across. Updates to the resource allocation instance reflect this data transfer. After successful data transfer, a data reference returns to the data management object, where an instance of a data object is created with this reference and returned by the node.

**Execute Job**

Figure 3.19 shows the interactions associated with a service execution. If a service does not require any input files, then the resource allocation node, directly connects to the execute service node, bypassing the data upload process.
A resource allocation and any data objects (if applicable) pass to an instance of a job service node which, when executed, submits the job submission details to the Job management object. The Job management object invokes the Grid job service, which in turn creates a new instance of a job. The workflow passes the data references to the Grid server to determine which data items from the data service are required. The job then executes, with the Job management object polling the job service to determine its progress. Upon successful completion of a job execution, the Data Service manages the outputs of the service and returns the references to the data.
Mapping Grid Functionality to Existing Scientific Workflow Systems

management object. Each output results in a data object instance to be returned by the execute job node.

**Download Data**

Figure 3.20 shows the interactions required for downloading a data item located on the Grid server. Either an execute job or upload data node produces a data reference, taken directly from the aforementioned nodes in a workflow or in a new workflow using a saved data item, to determine the data item to download.

---

**Figure 3.20: Interactions for downloading the results from a service**
The download data node does not necessarily represent the end of a workflow; therefore, the node returns the resource allocation for use with other Grid nodes further on in the workflow if required.

3.4.5 R-DTE Implementation

For comparison with the single node mapping approach, the R-DTE based framework implementation also uses InforSense and GRIA. Extensions to the administration interfaces include parameters for specifying data and job service endpoints, written to a file in a configuration folder located in the GRIA plug-in (represented in the bottom layer of the Grid handler design). The execute job nodes access these values upon instantiation from the workflow client, allowing the workflow author to choose a set of endpoints.

The resource allocation node in the InforSense implementation has no synchronization port by default, however the InforSense scripting framework allows for the addition of these ports to any node. This node implementation uses the same resource allocation parameters and methodology as the single Grid node, but returns an allocation model object as shown in Figure 3.21.
The upload data node includes a drop down list parameter, populated with data service endpoints at node instantiation. It takes a file reference and returns both a data reference to the file uploaded to the GRIA server and the updated resource allocation model.

The execute job node implementation generally uses the same job execution methods and parameters as the single node approach, including the ports and methods for handling the resource allocation propagated through the node. When dynamically adding ports to the node, however, only ports (of data reference types) are generated, because the dynamic parameters governing the data upload and download operations have been moved to their subsequent nodes.

The download data node uses the same parameters as the single node approach for determining the download location. The node also returns a file reference and the updated resource allocation model for further processing.
The data reference and resource allocation object implementations use the InforSense model framework. The model framework saves and persists data references and resource allocations in the InforSense Userspace. A workflow author can reuse these saved data reference and resource allocation Userspace items in a new workflow as a node. Using specialized models and types, the flow of data through GRIA workflows can be controlled and prevent incompatibilities between similar types, i.e. GRIA data references being specified where file references are required and vice versa.

3.4.6 SIMDAT Case Study: Updated Crash Simulation

An updated SIMDAT automotive crash simulation scenario tests the performance and usability of the R-DTE nodes in comparison to the single node mapping.

Updated Crash Simulation Environment

Apart from the InforSense GRIA plug-in, this scenario uses the same environment and technologies used in the previous automotive crash simulation scenario. The new workflows use nodes provided by the GRIA plug-in based on the R-DTE node mapping design.

Updated Crash Simulation Scenario

The first noticeable difference in the R-DTE implementation is the overall workflow size. More nodes are required in a workflow to perform a simple
Grid execution process, as shown in Figure 3.22. By separating the upload and download functions from the service execution, multiple crash simulations can run with different arguments, without having to upload the same input data each time.

Figure 3.22: Updated Crash Simulation Workflow using the RDTE Approach. More nodes are required for each service execution, but the performance and usability are improved

Scenario Evaluation

The Data Reference Model allows a workflow to access an item of data uploaded by another workflow, again saving on execution time and cost. Moving the data and job service endpoints to the administrator level improves security by ensuring that only authorized Grid servers can send and access data. Further comparisons between the performance of the R-DTE and the single node mapping are described later in this chapter.

The updated Crash Simulation Scenario also identifies a number of problems related to the use of the R-DTE implementation. First, the underlying crash
simulation and model combining applications update a number of times through the course of the scenario definition and implementation. This regular change in the number and types of inputs and outputs leads to workflow breakage and failure. A related, key problem is that the workflow author must know the exact inputs, outputs and arguments for each service. As experienced in the SIMDAT project, the workflow author does not necessarily know these details, making it increasingly difficult for them to compose and utilize services. In order to overcome this particular issue, metadata or provenance data for each service can help to describe required input and output details. Metadata can help workflow authors understand the changes to service interfaces, how to adjust workflows accordingly, and how to discover and compose new services, leading to quicker workflow turnaround. The crash simulation scenario also flags an issue related to the resource allocation process. The current method of resource allocation requires a workflow author to know or guess exactly how many resources the workflow will consume before execution. With the updated crash compatibility scenario, however, a domain expert may run a workflow multiple times until the simulation service produces suitable results. One way around the problem of authors inability to estimate resource consumption, is to allocate the maximum resource allocations for the worst-case scenario and receive reimbursements for any unused resources. Unfortunately, the reimbursements in this case are less than the initial allocation cost and thus result in unnecessary losses. Adopting a pay-per-usage model, which charges for every resource consumed and provides a daily/monthly/yearly
Mapping Grid Functionality to Existing Scientific Workflow Systems

bill, helps to overcome this problem. The next form of node mapping adopts such a model, completely removing the resource allocation process from the workflow system.

3.5 DTE-M Mapping

The DTE-M (Data Transfer, Execute job and Metadata) mapping is a refinement of the single and R-DTE mappings. The particular focus is metadata and removal of the resource allocation process from the workflow level, to streamline the service execution process. Such a node mapping approach is similar to the mappings described in [104] and [2].

3.5.1 DTE-M Grid Handler

Figure 3.23 shows the Grid handler definition for the DTE-M approach. Compared to the R-DTE Grid handler, the DTE-M handler has an additional metadata support interface and removes the resource allocation based components from the node definition and support layers.
3.5.2 DTE-M Nodes and Functionality

This section outlines the node definitions and functionality support of the DTE-M approach. Note that the Upload Data, Download Data and Data Reference Object nodes remain unchanged from the R-DTE mapping, except for the removal of the resource allocation ports and processes.

**Resource Allocation**

To avoid the resource allocation drawbacks outlined in the Automotive case studies, the DTE-M mapping employs a pay-per-usage billing scheme. Administrators can use the Grid client in tandem with the workflow based administrative tools node to support the resource allocation functionality. The Grid DTE-M handler does not require resource allocation associated nodes and objects.
Execute Job

The parameters that define a DTE-M based execute job node reflect the metadata made available by the Grid server. The node configuration can follow two steps. The first step, service discovery, requires the Grid server to provide a mechanism for discovering its hosted Grid services. For basic node configuration, the metadata for each service should include input/output information allowing data reference ports to be automatically determined. The node interface can also extract and utilize additional information included within the metadata, such as descriptions and service arguments. Where metadata is not available, the node must be manually configurable as described for the single and R-DTE node mappings. To support the metadata handling the Grid handler provides an additional metadata support object, described later in this section. Note that metadata handling, involves interactions with the Grid service at author time, which requires service calls via the workflow server.

3.5.3 DTE-M Support Interfaces

The helper interfaces for the DTE-M mapping extend the R-DTE approach with support for additional metadata functionalities required to parameterize the DTE-M execute job node. Note that the Data management, security and administration helper interfaces remain unchanged from the R-DTE mapping.
Job Management

Given that the DTE-M mapping requires the retrieval of extra metadata information during workflow authoring, job management helper object extensions support further calls to the Grid server. These extra operations include retrieving both a list of available services based on a job service endpoint, and the metadata for each service. The parsing of metadata can be handled by a separate metadata helper object that works in tandem with this component, as shown in the interactions for this mapping.

Metadata

The metadata object contains the parsers required to process the results returned by the job management object during service discovery and metadata retrieval. Full node utilization implies the service metadata should ideally contain the following information:

- **Ports**: A reference name for each port is required, to identify the data passed to and returned from the service. Metadata can provide, for each port, type information to perform extra error checking, i.e. check if the correct type of file is passing to the service. Where possible, the metadata should also indicate whether an input port is compulsory or optional, in order to provide better error handling and support the authoring process.
• **Arguments:** Like the ports, each argument should have a name and type for mapping to parameters. If possible, a default value, range of values or bounds (depending on the type of argument) should also be provided and added to the node parameters, enabling the user to execute a service without having to set parameters manually each time.

• **Descriptions:** The metadata can provide optional descriptions that explain the operation of the service or provide further details about the inputs, outputs and arguments. Although the descriptions are not required to parameterize the node, they can aid the workflow author in terms of workflow composition.

Grid servers usually provide metadata in a structured format, such as XML, for easier parsing.

**Metadata Utilization Design**

The pseudo code in Figure 3.24, Figure 3.25 and Figure 3.26 suggests potential designs for the execute job node, job management support interface and a new metadata support interface for handling application metadata.
EXECUTE JOB

CALL JobServiceMgmt.getJobServiceList RETURNING jobServiceList
Select jobService from jobServiceList RETURNING jobServiceURL
CALL JobServiceMgmt.getAppList with jobServiceURL RETURNING appList
Select application from appList RETURNING metadata and serviceEndpoint

Save jobserviceURL and serviceEndpoint
Clear node inputs, parameters and outputs

CALL metadata.getInputs RETURNING inputMetadata

IF metadata has inputs
    FOR EACH inputMetadata item
        CALL item.getName RETURNING name
        CALL node.addInput with name
    ENDEACH
ENDIF

CALL metadata.getOutputs RETURNING outputMetadata

FOR EACH outputMetadata item
    CALL item.getName RETURNING name
    CALL node.addOutput with name
ENDEACH

CALL metadata.getParameters RETURNING paramsMetadata

IF metadata has parameters
    FOR EACH paramsMetadata item
        CALL item.getName RETURNING name
        CALL item.getType RETURNING type
        CALL node.addParameter with name and type
    ENDEACH
ENDIF

Figure 3.24: Pseudo code design for DTE Execute job node that includes extra methods for metadata handling
Figure 3.25: Pseudo code design for updated Job management interface that handles application metadata
Figure 3.26: Pseudo code design for metadata support interface

3.5.4 DTE-M Interactions

The following interactions occur between the components in the DTE-M mapping during configuration, authoring and execution. Given the removal of the resource allocation process, the DTE-M mapping interactions are simpler than in the R-DTE mapping.
Authoring Interactions

To support a metadata-based node mapping approach, calls to the Grid interfaces may be necessary during preparation time. Figure 3.27 shows the interactions between components across the different layers to map the execute job node to a selected service.

![Diagram showing interactions between components](image)

**Figure 3.27: Interactions for selecting an application from a Grid server**

A call initially made from the node interface selects applications available on the Grid server. The Job Management interface retrieves a list of applications from the job service and presents it to the workflow author. Selecting an application makes another call to retrieve its associated metadata. The
metadata helper object parses the returned metadata and sends the values required to configure the node to the Job Management object. The node interface presents the metadata to the workflow author, and upon service confirmation, the node is parameterized accordingly.

### Upload Data

Figure 3.28 identifies the interactions for uploading a file to the Grid server in accordance with DTE-M mapping. A similar process takes place for the R-DTE mapping but with resource allocation propagation.
Execute Job

Figure 3.29 shows the process for executing a job. The node invokes a job with the data references passed to it from a data model object. Like the R-DTE based implementation, the node returns a data model for each output upon successful execution.

Download Data

Figure 3.30 shows the process for copying a file from the grid data server to the workflow server’s local environment using the DTE-M approach. The
reference is taken from a data model returned by an execute job, upload node or persisted data model node.

![Diagram of workflow client, grid handler node definition, grid handler support interfaces, and grid interfaces]  

Figure 3.30: Interactions for downloading results using the DTE-M mapping

### 3.5.5 DTE-M Implementation

The DTE-M implementation extends the R-DTE implementation for the InforSense workflow system accessing GRIA. Changes to the Execute Job node support the metadata and service discovery functions provided by the GRIA middleware. Rather than manually entering the service details and manually determining the ports, the DTE-M approach automates this
information by utilizing service metadata. Figure 3.31 shows the custom parameter (implemented in Java Swing) for finding services and viewing metadata from the Execute job node.

Figure 3.31: Interface for selecting a GRIA service supporting metadata utilization

Selecting a service endpoint from a drop down list (populated by the administrator, in similar fashion to the R-DTE Execute Job node) invokes a web service call to the GRIA server to retrieve a list of applications. The Metadata support interface parses the service list, with the node displaying the services in another drop down format. Selecting an application invokes
another web service call to retrieve the metadata file for the selected service. The information displayed includes the names and types for the service inputs/outputs and a description of the service’s functionality. Note that the node caches each call’s results for the duration of the workflow life, to reduce the number of service calls. Confirming the service selection creates and persists the parameter information, allowing it to be stored with the workflow. The ports are dynamically added/removed on the node to reflect the inputs and outputs specified in the service metadata. The node also gives each port a reference name based on the names specified in the metadata.

3.5.6 SIMDAT Case Studies: Master Sequence Analysis Pipeline

The MSAP/Drugability [10] scenario defined in the Pharmaceutical domain as part of the SIMDAT project involves the execution and co-ordination of high performance analysis tools to determine how effective a protein is for developing new drugs.

MSAP Scenario Environment

The MSAP scenario environment consists of an InforSense v4 installation at a Top 10 Pharmaceutical company. This particular deployment consists of a separate client/server setup with a single server installation and multiple clients distributed amongst a research group. The MSAP workflow implemented in InforSense access a combination of in-house services
deployed as command line based applications and GRIA v5 services. The workflow also invokes services deployed at an external UK based drug-discovery firm and a Belgian academic institute. Since all GRIA services are deployed using version 5, an InforSense GRIA v5 plug-in is also installed based on the DTE-M mapping.

**MSAP Scenario**

For more thorough analysis and to increase the potential chances for a good match, a range of different analysis tools are used and the results collected and analyzed by a workflow, as shown in Figure 3.32. Deployed tools reside locally, as Web services or Grid services. The tools deployed as Grid services are hosted by GRIA, and InforSense provides the workflow system with capabilities to handle all forms of services through Bio, Web and GRIA (DTE-M based) plug-ins. The MSAP/Drugability scenario combines the challenges associated with the Knowledge Services and Automotive scenarios in that it involves interoperating between different types of services and chaining together various Grid services.
Figure 3.32: MSAP using a combination of GRIA and local services based on the DTE-M approach

Given the high number of workflow branches defined to support the range of services, and the elaborate details at the workflow level, the workflow is deployed the InforSense portal. The portal interface encapsulates technical details in accordance with the Domain Expert user role, revealing only the parameters associated with the function of each analysis tool at the portal level. Figure 3.33 shows the portal interface for the MSAP.
Figure 3.33: Portal interface for MSAP workflow that hides the details of the workflow underneath

**Scenario Evaluation**

Removing the resource allocation process from the workflow system and using only three nodes to construct Grid based workflows markedly improves usability; it reduces the number of nodes required to execute a Grid service and follows a more logical set of operations. However, the one node per service methodology provides a better visual understanding and representation of the steps in a workflow. As shown in the interaction diagrams, removing the resource allocation functionality simplifies the node implementations. The metadata support also aids service discovery and quickens the workflow construction process. The MSAP solution also shows the DTE-M mapping allows seamless integration with other workflow nodes and other forms of services, facilitating higher-level deployment.
One drawback of the DTE-M approach is that it requires Grid server interaction during authoring time and because Grid server access occurs through the workflow server, metadata support may not be possible for some scientific workflow systems. Where this is the case, a metadata-less execution node is required, such as the one defined in the R-DTE mapping. Similarly, depending on the Grid server, it may not be possible to remove the resource allocation process from the workflow system.

### 3.6 Node Mappings Performance Evaluation

The various node mappings have been evaluated using case studies to identify their strength and weaknesses. In order to perform a fair evaluation, all tests use the same environment with as many variables as possible kept consistent. A standalone InforSense v4 installation was setup with GRIA v5 plug-ins for all three mappings to ensure uniformity. The workflows built using these plug-ins, connect to GRIA v5 image manipulation services running at a UK research centre and a university in Egypt. Both these environments were stable with consistent performance. All tests (and repetitions) were performed when network bandwidth provided minimal fluctuation in order to regulate data transfer times, with the same services and input files/parameters. The GRIA client is used to perform the native GRIA calls since this is the default client for performing GRIA service calls. This application calculates the time taken for the required methods to run,
allowing for comparisons between these results and the workflow execution times retrieved from the InforSense task manager. The following graphs focus on the performance of the single, R-DTE and DTE-M mappings. A number of performance tests measure the execution times for a range of operations and workflows.

3.6.1 Comparing Workflow Performance to Native Grid Calls

Table 3.1 and Figure 3.34 compare the relative upload times for the DTE-M mapping and the native GRIA calls, and measure the percentage difference between the two approaches. The experiments to produce these comparisons range over file sizes. The results show a marked increase in execution time for the DTE-M mapping, indicating the workflow overhead; however, this overhead has very little effect on the overall execution time, especially as the file sizes increase.
### Table 3.1: Comparison of workflow and native data upload times

<table>
<thead>
<tr>
<th>File Size (Kb)</th>
<th>Native (GRIA) (s)</th>
<th>InforSense (s)</th>
<th>% Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.5</td>
<td>1.6</td>
<td>260.4</td>
</tr>
<tr>
<td>1002</td>
<td>5.5</td>
<td>6.8</td>
<td>24.7</td>
</tr>
<tr>
<td>2002</td>
<td>11.0</td>
<td>11.7</td>
<td>7.01</td>
</tr>
<tr>
<td>3002</td>
<td>15.3</td>
<td>16.4</td>
<td>6.9</td>
</tr>
<tr>
<td>5122</td>
<td>26.9</td>
<td>29.2</td>
<td>8.3</td>
</tr>
<tr>
<td>10401</td>
<td>53.9</td>
<td>58.8</td>
<td>9.2</td>
</tr>
<tr>
<td>20980</td>
<td>101.9</td>
<td>109.1</td>
<td>7.0</td>
</tr>
<tr>
<td>52502</td>
<td>268.7</td>
<td>284.1</td>
<td>6.5</td>
</tr>
<tr>
<td>104113</td>
<td>516.9</td>
<td>541.8</td>
<td>4.8</td>
</tr>
</tbody>
</table>

Table 3.1 and Figure 3.34 compare the download times and differences for the DTE mapping and the native GRIA calls. The download experiments, like the upload, use a range of file sizes. Again, the workflow overhead is noticeable, but negligible.
Mapping Grid Functionality to Existing Scientific Workflow Systems

<table>
<thead>
<tr>
<th>File Size (Kb)</th>
<th>Native (GRIA) (s)</th>
<th>Workflow (s)</th>
<th>% Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.4</td>
<td>0.7</td>
<td>58.5</td>
</tr>
<tr>
<td>1002</td>
<td>5.4</td>
<td>5.7</td>
<td>5.6</td>
</tr>
<tr>
<td>2002</td>
<td>10.4</td>
<td>11.5</td>
<td>10.8</td>
</tr>
<tr>
<td>3002</td>
<td>15.1</td>
<td>16.2</td>
<td>7.3</td>
</tr>
<tr>
<td>5122</td>
<td>25.5</td>
<td>28.4</td>
<td>11.4</td>
</tr>
<tr>
<td>10401</td>
<td>50.9</td>
<td>55.7</td>
<td>9.3</td>
</tr>
<tr>
<td>20980</td>
<td>102.8</td>
<td>106.0</td>
<td>3.1</td>
</tr>
<tr>
<td>52502</td>
<td>257.1</td>
<td>264.8</td>
<td>3.0</td>
</tr>
<tr>
<td>104113</td>
<td>522.9</td>
<td>534.7</td>
<td>2.3</td>
</tr>
</tbody>
</table>

Table 3.2: Comparison of workflow and native data download times

Figure 3.35: Comparison of workflow and native data download times

Table 3.3 and Figure 3.36 compare the execution of a single service using the DTE-M mapping and native GRIA calls as input file sizes vary. The workflow overhead is more considerable compared to the upload and
download experiments, caused primarily by the polling mechanisms and log recording performed by the node, in addition to service initiation overheads.

<table>
<thead>
<tr>
<th>File Size (Kb)</th>
<th>Native (s)</th>
<th>Workflow (s)</th>
<th>% Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>18.687</td>
<td>23.093</td>
<td>23.577</td>
</tr>
<tr>
<td>1002</td>
<td>19.672</td>
<td>28.440</td>
<td>44.573</td>
</tr>
<tr>
<td>2002</td>
<td>26.010</td>
<td>30.207</td>
<td>16.133</td>
</tr>
<tr>
<td>3002</td>
<td>29.219</td>
<td>33.740</td>
<td>15.474</td>
</tr>
<tr>
<td>5122</td>
<td>34.876</td>
<td>39.667</td>
<td>13.736</td>
</tr>
<tr>
<td>10401</td>
<td>52.911</td>
<td>56.577</td>
<td>6.928</td>
</tr>
<tr>
<td>20980</td>
<td>89.078</td>
<td>92.743</td>
<td>4.114</td>
</tr>
<tr>
<td>52502</td>
<td>215.823</td>
<td>221.953</td>
<td>2.841</td>
</tr>
</tbody>
</table>

Table 3.3: Comparison of workflow and native job execution times

These experiments detect noticeable workflow overhead for each of the operations when compared to native GRIA service calls. Nevertheless, the
overhead affects are practically negligible, especially where larger data sizes or longer running jobs are involved.

### 3.6.2 Comparing the Performance of Node Mappings

The following experiments compare the performance of each of the node mappings. Each of these experiments use a number of workflows, ranging from a single service to five services chained together. Different initial input file sizes were used with each of the workflows; 1Mb, 5Mb and 10Mb. Table 3.4 and Figure 3.37 shows the performance of workflows that are implemented using the single (grey), R-DTE (orange) and DTE-M (blue) approaches. Each service accessed from a workflow is hosted on a different GRIA server and therefore requires data upload and download processes. Note that the execution times for the R-DTE and DTE-M workflows are marginally higher when compared to the single node workflows, due to the workflow overheads of invoking multiple nodes instead of one for each service invocation. This overhead is minimal and does not have a great bearing on the overall performance.
<table>
<thead>
<tr>
<th>No. of Nodes</th>
<th>1Mb</th>
<th>5Mb</th>
<th>10Mb</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sgle</td>
<td>R-DTE</td>
<td>DTE-M</td>
</tr>
<tr>
<td>1</td>
<td>39.8</td>
<td>45.3</td>
<td>44.3</td>
</tr>
<tr>
<td>2</td>
<td>111.2</td>
<td>118.6</td>
<td>115.6</td>
</tr>
<tr>
<td>3</td>
<td>165.7</td>
<td>171.6</td>
<td>167.6</td>
</tr>
<tr>
<td>4</td>
<td>224.4</td>
<td>230.5</td>
<td>227.5</td>
</tr>
<tr>
<td>5</td>
<td>269.4</td>
<td>280.2</td>
<td>275.2</td>
</tr>
</tbody>
</table>

Table 3.4: Execution times for each node mapping with different file sizes and number of services, each running on a different GRIA server.

Figure 3.37: Single node, R-DTE and DTE-M node mapping execution times for workflows consisting of 1-5 services, running on different servers with 1, 5 and 10 Mb input files, respectively.
Table 3.5 and Figure 3.38 show the performance of each of the workflows with all services deployed on the same GRIA server. For the R-DTE and DTE-M approaches, data upload and download nodes are not required between each service, thus the outputs of each service are directly referenced and used by the input of the next service. The removal of unnecessary data transfer operations in the R-DTE and DTE-M mappings vastly improves the performance compared to the single node mappings, especially where more services are concerned.

<table>
<thead>
<tr>
<th>No. of Nodes</th>
<th>1Mb</th>
<th>5Mb</th>
<th>10Mb</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sgle R-DTE DTE-M</td>
<td>Sgle R-DTE DTE-M</td>
<td>Sgle R-DTE DTE-M</td>
</tr>
<tr>
<td>1</td>
<td>39.8 36.3 34.3</td>
<td>83.8 80.2 80.2</td>
<td>158.7 158.6 158.6</td>
</tr>
<tr>
<td>2</td>
<td>87.4 76.3 70.3</td>
<td>198.2 166.2 163.2</td>
<td>405.2 305.1 295.1</td>
</tr>
<tr>
<td>3</td>
<td>139.2 111.2 101.2</td>
<td>277.7 213.9 207.9</td>
<td>554.1 362.3 357.3</td>
</tr>
<tr>
<td>4</td>
<td>183.1 150.0 145.0</td>
<td>390.8 300.9 294.9</td>
<td>788.1 504.3 500.3</td>
</tr>
<tr>
<td>5</td>
<td>231.0 182.2 176.2</td>
<td>456.7 341.3 336.3</td>
<td>924.1 561.0 556.0</td>
</tr>
</tbody>
</table>

Table 3.5: Execution times for each node mapping with different file sizes and number of services, each running on the same GRIA server.
3.6.3 Comparing Component Interactions

The single node mapping performs all the functions from one node, which reduces the number of components and their interactions, but this non-modularized approach makes extensibility and maintainability difficult. The R-DTE mapping presents a more modularized approach, addressing some of the issues described that are associated with the single node mapping. Unfortunately, the number of components and interactions increases significantly, complicating the implementation. The interactions for the DTE-M mapping strike a balance between the other two mappings, reducing the
complexity associated with the R-DTE approach, while maintaining a level of modularity that the single node approach lacks.

3.7 Summary and Conclusions

This chapter identifies three potential forms of node mapping (single node, R-DTE and DTE-M) and a Grid handler design for each to address the access of Grid functionality via workflow systems. Mapping Grid functionality to workflow nodes is a non-trivial process given the remote nature of Grid services and the broad range of functionality to be supported from the workflow system. Previous work in Grid-workflow mappings [2][104][109] failed to identify the performance of node mappings, components within a layered Grid-workflow architecture and the interactions between them--key parts of the design and key to understanding the integration process. Analyzing the component-level interactions can also determine the amount of work required and complexities associated with each node mapping.

A single node approach lends itself to intuitive workflow since a single node represents a single service. However, with this form of node mapping, the node interface itself becomes more complicated. The parameters are confined within a single interface and the stateful features associated with Grid services are not supported, leading to erroneous and expensive data transfers. Separating Grid functionality addresses the issues associated with
the single node approach. Separation makes it possible to persist data references and reuse data already transferred to Grid servers, which is paramount for large data sizes as those associated with high-performance, computationally intensive jobs. Using multiple nodes to execute a service will of course lead to bigger workflows, but combinations of these mappings are feasible depending on the Grid middleware and workflow system used; while a R-DTE-M mapping can utilize both resource allocation and metadata functionality, a DTE mapping lacks the resource allocation and metadata support. Regardless of the node mapping form employed, only advanced workflow features, such as authoring time interactions and pre-authoring administrative configuration take full advantage of Grid middleware.

This chapter only covers and demonstrates the integration for a single workflow system and server. In order to test the framework’s flexibility, other Grid middleware and Workflow Systems must be explored.
Chapter 4
Framework Extensions for Addressing Grid Heterogeneity

The previous chapter introduces and defines a generic framework for accessing and invoking remote Grid services. The modularity of the architecture described in Chapter 1, allows implementations of Grid and workflow technologies to be interchanged, however the framework does not address heterogeneity. Heterogeneity is an important concept to address with challenges focusing on usability in terms of workflow composition and performance. As shown in the case studies presented in this chapter, some scenarios require the use of services deployed using different Grid middleware. Although it is possible for workflow systems to support each middleware using the framework described in the previous chapter, when more than Grid middleware is present this can lead to issues such as library clashes, increased maintenance and different look-and-feels.

This chapter extends the Grid-workflow framework to support two forms of Grid heterogeneity, the Gateway and Adaptive API approaches, outlining the advantages and disadvantages of each approach with respect to usability and performance. DockFlow [8][111], a collaborative protein-docking scenario outlined in the BRIDGE project, tests and evaluates the
heterogeneity framework extensions. The DockFlow scenario also emphasizes the importance of service location in Grid workflow design, and consequently examines and evaluates various forms of service deployment and access methods.

### 4.1 Grid Heterogeneity Requirements

Generally, Grid services required to build a workflow applications are not always hosted on the same Grid middleware. Each middleware has its own set advantages and disadvantages as well as being suited for particular types of applications and environments. As with workflow interoperability where adhering to a single standard has many pitfalls, adhering to a Grid middleware standard or adopting only one middleware in a VO context, greatly reduces the functionality available, and presents a number of challenges in its own right. By achieving Grid interoperability and heterogeneity, it is possible to make full use of the different tools available and to get the most out of the technology with minimal affect on usability and loss of functionality.

The overall aim of this chapter is to extend the framework identified previously to orchestrate Grid services deployed with different Grid middleware using Grid heterogeneity. The focus is on usability, both in terms of integration (i.e. service access and invocation) as well as orchestration of services deployed on different middleware as simply and efficiently as
possible. Grid heterogeneity and interoperability through workflows raises a number of challenges:

- **Services Access:** Each Grid middleware has a different method for deploying services, i.e. with GRIA an application script handles the inputs, outputs and parameters and can include other functions, with GLOBUS/CAGrid web service stubs are automatically generated and the inputs, outputs and parameter objects and types must be defined. These deployment methods affect how the execution of services and the passing of data to and from services. For each middleware different operations are defined for performing the execution, handling the data, managing security and so forth. In order to perform these operations, the Grid middleware provides an API for building clients to access these services (as described in the previous chapter) and has a major effect on the integration process.

- **Data Transfer:** When more than one Grid middleware is involved, transferring data becomes an issue as each middleware has its own definition and interfaces for data items stored on the Grid server. These discrepancies affect the node definitions and the design of the objects transferred between them.

- **Minimising Loss of Functionality:** As each Grid middleware has its own strengths, it is crucial to preserve the advantages they bring
when trying to achieve Grid heterogeneity. It is also important to find a necessary medium between functionality and usability. Supporting more functions allows for the execution of a wider range of services and allows for more flexibility in order to meet user requirements. However, the more Grid functionality supported by an SOA based workflow the more complex the node definition process can become, thus reducing usability when it comes to composing Grid based workflows.

4.2 Approaches for Addressing Heterogeneity

The standard approach for addressing Grid heterogeneity, within the context of a Grid-workflow framework defined in the previous chapter, involves adding separate Grid extensions (i.e. separate Grid handlers) to the workflow system for every Grid middleware supported. The advantage of using such an approach is that it supports full functionality (in terms of what is required from a workflow perspective) as each handler can be defined independently from other Grid handlers. Using this individual integrative approach, direct communication with the middleware is achievable as well as direct access to services hosted. Direct access potentially reduces the execution time of Grid based workflows as the overheads are minimised. However, integrating middleware separately can be problematic from a development perspective. Each handler design is separate and varies depending on the functionality required. Each handler implementation is also separate and updated
individually, which can have a great affect on the development time. Employing a separate integration approach also leads to nodes with a different look and feel given that each middleware provides different types of functionality and access methods. Note that this individual integrative approach to Grid heterogeneity only works at a client level and does not achieve actual middleware interoperability. Furthermore, not all workflow systems have independent classpaths, which can lead to clashes between library versions for each of the middleware interfaces.

Two interoperability approaches, the Gateway and Adaptive API approaches, are potential solutions for overcoming or reducing the aforementioned issues [107].

### 4.3 Gateway Approach

Within the context of a Gateway approach, a workflow system only interacts with one Grid middleware, which acts as a proxy to services hosted on other middleware. This proxy middleware is responsible for generating the files required by any secondary middleware for invoking services as well as relaying service calls. Essentially, a proxy middleware acts as the access interface to further Grid systems. As the proxy Grid middleware is responsible for handling the interoperability, it needs extensions for supporting communication with other Grid data and job services, and mirroring the metadata for each service on the proxy server. A Grid handler
is then built on the proxy middleware interfaces and access to other middleware is handled behind the scenes.

The advantages associated with utilizing a Gateway approach are that no or very few changes would have to be made to a workflow Grid handler to support Grid heterogeneity, and a common Grid node interface is always presented to the workflow author, regardless of the number of middleware supported.

4.3.1 Gateway Approach-Based Architecture

Figure 4.1 below shows a high level layered architecture for supporting the Gateway approach using the Grid-workflow architectures defined in the previous chapters.
Given that the Grid handler only accesses one Grid handler, any of the node mapping designs and their associated Grid handlers described in the previous chapter are applicable here. Throughout this chapter, the architectures and components use the DTE-M node mapping design, since the single node mapping is inadequate for addressing the interactions and challenges associated with heterogeneity (through the lack of state based...
functionality), while the R-DTE mapping further complicates the processes through the introduction of resource allocation mechanisms.

4.3.2 Gateway Approach Interactions

The following interactions extend the DTE-M component definitions and interactions with support for the Gateway Approach.

Upload Data

The upload process interactions for the Gateway approach follow a similar pattern to the upload process for the DTE-M components but with an additional layer, that identifies the transfer of data from the proxy Grid server to another. Once the data transfer to the secondary Grid server is completed, the proxy Grid server returns a reference to the Execute job node. Figure 4.2 shows the Upload data process for the Gateway approach.
Figure 4.2: Uploading data to a secondary server via a proxy

**Execute Job**

Figure 4.3 shows the interactions for invoking a Grid service hosted on a secondary Grid server.
When a service is invoked by the workflow system an instance of a job is created on the proxy Grid server. The proxy Grid server submits the required information to perform an execution on the secondary Grid server along with any arguments and data references. The secondary server invokes an instance of the service using the data references. When the Execute Job node submits a polling request to the proxy job instance, the proxy server forwards the request to the secondary server. After the job has completed,
the secondary Grid server manages the output of the job via the data service. For each output, the proxy server mirrors the data reference, passing it the workflow system for use with the next node.

**Download Data**

Data items downloaded from the secondary Grid server follow the interactions shown in Figure 4.4. A request made from a node to download data from the secondary Grid server, goes via the proxy server. The second Grid server copies the data to the proxy Grid server. Upon completion, the data can then be downloaded, and a file reference generated, by the workflow system.
4.3.3 Gateway Approach Implementation

To test the framework for supporting the Gateway approach, the Grid environment deployed in the BRIDGE project consisting of GRIA and GOS middleware is used. Wang in [107] describes the GRIA extensions for supporting interoperability with the GOS middleware. A GRIA proxy server communicates with GOS Servers and invokes the services hosted on them.
To support the Gateway implementation, the workflow system accesses the GRIA servers using the DTE-M based GRIA plug-in for InforSense as described in the previous chapter. Only minimal changes to the GRIA handler are required in order for it to be compatible with the GRIA extensions to support access to the GOS middleware. For each service deployed on GOS middleware, a wrapper script on the GRIA proxy server deploys a job stub for the corresponding GOS service and provides metadata for service discovery and node parameterization.

The following process indicates how an InforSense workflow, with Gateway approach extensions, can access GOS services. Using the Execute Service node, a call made to the GRIA job service endpoint lists available GRIA and GOS services. Selecting a GOS service performs a call to retrieve the metadata provided on the GRIA server allowing for the dynamic addition of node ports and parameters. The workflow submits any data required by the GOS service to the GRIA server and subsequently the GOS server using Upload Data nodes. Invoking the Execute Service node generates a JSDL file required by GOS consisting of the GOS service endpoint, a reference to the input data, now copied onto the GOS server, and reference names for the output data (taken from the port names of the Execute Job node). GRIA submits the JSDL file to the GRIA job service to the GOS service. The GOS service is then invoked using the input data referenced with the job progress relayed back to the GRIA application wrapper, allowing the workflow to access the logs. Once the job is complete, GOS submits the results to the
GRIA Data Service, which returns the references to the Execute Service node. Download Data nodes can then be used to copy the data from the GRIA server, which in turn pulls the data from the GOS server.

### 4.3.4 BRIDGE Case Study: DockFlow – Protein Docking

The DockFlow scenario, identified as part of the BRIDGE project, is suitable for testing the Gateway approach framework, as it requires services distributed across GRIA and GOS middleware. DockFlow is a virtual screening environment for protein docking that integrates four Grid-based protein docking tools executing using two types of Grid middleware in different locations. Protein docking is a compute intensive application that aims to find small molecules that can modulate the actions of proteins and the interactions between them. Dockings are scored based on a set of parameters for each compound and filtered for further analyses. A number of algorithms and computational tools exist for addressing this protein-docking problem. However, due to the different assumptions used by each docking algorithm, the results can vary from one tool to the next. These variations, range from minimal deviations to drastic discrepancies even for the same ligand-receptor pairs. DockFlow aims to collate the results from different docking tools and run a scoring algorithm to compare the results from the different scoring methods, identifying points of commonality.
**Protein Docking Environment**

Figure 4.5 shows the schematic for the DockFlow concept. An InforSense v4 workflow system deployed in the UK accesses two protein-docking tools, AutoDock and FlexX, deployed at a research institute in Germany. GRIA v5 is used to deploy these services since it supports JSDL job submission for supporting the Gateway approach and has improved performance using REST web services as opposed to SOAP based messages. The DOCK and GAsDOCK docking tools are deployed at an institute in the People’s Republic of China using GOS middleware. The GOS deployment consists of a head node and distributed slave nodes. The GOS resource management system schedules each job invocation. The workflow saves the results from each docking tool in a database in Germany and runs a Python based scoring tool over the collected results. Note that the InforSense installation includes a GRIA v5 plug-in based on the DTE-M mapping to support the extra GRIA v5 features required for the Gateway approach.
Protein Docking Scenario

DockFlow consists of four sub-workflows, which coordinate the steps required to invoke each of the docking tools. Each sub-workflow structure for invoking each of the tools follows a similar process. The workflow fetches data from ligand and compound databases according to pre-defined user queries. The data then undergoes pre-processing to prepare and format the data for submission to the docking tools. The workflow then submits the data to the Grid-based implementation of the docking tool. Post-processing is applied to the outputs of each tool before they are stored in a common result database from which the different results can be compared, analysed, and visualized as shown in Figure 4.6.
Case Study Evaluation

Using the gateway approach, services running on both GRIA and GOS middleware were successfully accessed, achieving heterogeneity to solve a real life collaborative scenario. The Gateway approach is the simplest form of Grid interoperability from a workflow development perspective with the interoperability mechanisms pushed to the Grid layers. Minor development is required at the workflow level in order to support access to multiple Grid middleware. From an authoring perspective, the Gateway approach providing a single, common interface to all supported Grid middleware thus improving usability since there is no need for explicit knowledge about supported Grid middleware.
The gateway approach however has a number of limitations. Additional proxy servers are required to handle the data transfer and service interaction processes. These proxy servers must be synchronised with the servers they access in terms of service metadata (i.e. metadata must mirror the changes to service interfaces), application wrappers, as well as server patches and updates. Furthermore, adding an extra server increases the risk of failure, i.e. if the proxy server fails, then a host of services up and running on a secondary server will not be accessible. Adding a proxy server also increases the number of overheads when executing a service. For each data item transferred between the workflow and a secondary Grid server, the data must first travel to and from the proxy server. This can vary the data transfer time greatly as each upload and download process occurs twice between the proxy and secondary Grid servers. Similarly, for each service call, which includes initialising jobs and polling running jobs, the information passes between both servers and the workflow. The proxy Grid server is also responsible for preparing the information required by the secondary server, adding another overhead to the mix.

4.4 Adaptive API Approach

The Adaptive API approach towards heterogeneity attempts to address some of the shortcomings of the Gateway approach. With an Adaptive API approach, a set of workflow nodes for accessing a Grid Service need to be implemented using a common API that interfaces to all supported systems,
thus allowing each Grid middleware to be accessed directly. A Grid handler for accessing Grid servers based on the Adaptive API approach can be implemented using any or a combination of the node mapping designs described in the previous methods. The basis for such a Grid handler however builds upon abstract methods and common shared parameters.

4.4.1 Adaptive API Approach-Based Architecture

Figure 4.7 shows a high level layered architecture for supporting the Adaptive API approach.
Figure 4.7: Layered architecture for supporting the Adaptive API approach

4.4.2 Adaptive API Grid Handler

Figure 4.8 below shows a design for the Adaptive API Grid handler based on the DTE-M node mapping. The largely modified interfaces are in blue.
Within an adaptive API based Grid handler, the main changes are made to the data and job management support interfaces which are to now support abstract methods rather than being tied to a particular Grid middleware implementation. Note that this interface abstraction may affect the interfaces presented in each of the node definitions. An abstract data model replaces a middleware specific data model and can now refer to data items stored on any of the supported Grid servers. To support the abstract component definitions, the Grid handler must package the Adaptive API based libraries and configuration files as defined in the bottom layer.

4.4.3 Adaptive API Nodes

The following design considerations must be taken into account when defining each node based on the Adaptive API:
Abstract Data Model

The abstract data model has to support multiple data service interfaces for different Grid middleware. In order to aid workflow authoring, it should be possible for a data reference generated on one Grid middleware to be passed to another data service on another Grid middleware (if communication is possible between them), with the API handling the exchange. In order to support this operation, the abstract data model must persist a form of identification, which dictates the middleware used and the access parameters specific to that middleware.

Upload Data

Different middleware require different parameters for uploading data to their data service. The API used determines how these parameters are handled. Either a single set of parameters are provided and the API handles the translation or conversion to the parameters required by each data service or the parameters specific to each data service are handled by the node itself. The latter approach is more commonly used and while it may require more work (parameters associated with all supported Grid middleware need to be included in the node interface), it provides more control compared to the former approach. One way to limit the parameters shown to the workflow author and limit confusion is by using dynamic parameters, where the parameters on a node change to reflect the Grid server selected. Of course, supporting this functionality is only possible if the workflow system supports
dynamic node interface definitions. In some cases, grouping of parameters can also indicate which parameters are associated with each middleware.

**Execute Job**

The Execute Job node definition must support the abstract data reference model associated with the Adaptive API. Like the upload data node, the interfaces for the Execute Job node may also vary depending on the job services accessed. Dynamic parameters may be used here to differentiate between the Grid middleware systems used. However, the design for the job execution node, must also consider application metadata. Metadata for each service may vary in terms of content as well as structure, and in some cases, no application metadata may be available from a Grid server. The metadata component in the framework also needs extending to include different parsers to handle different forms of metadata. Regardless of the form of metadata, the following information would always be required for job execution: ports, job service and job URI.

**Download Data**

The download data implementation must also support an abstract data model as an input. In terms of parameters for defining where a file is copied to, no changes need to be made from the DTE-M mapping. An instance of the abstract data model should contain all the information required by the Adaptive API to copy the data to the workflow system.
4.4.4 Adaptive API Support Interfaces

The Support Interfaces for the Grid handler must also change with the introduction of the adaptive API. Supporting the Adaptive API affects the following DTE-M based interfaces.

Security

The security settings required by each middleware must be supported. For example, the GOS middleware uses FTP for managing data, thus usernames and passwords are required to interact with GOS FTP sites where as GRIA uses keystores. The workflow needs to support and persist the security settings when necessary.

Administration and Configuration

Like the security helper interface, the administration and configuration object components must be extended to support each middleware. Changes from the DTE-M mapping include a way to enter security settings, endpoints and associated validation for each Grid middleware supported. For example, the administration interfaces should allow for the specification of FTP sites along with their associated username and password pairs. The workflow system should validate these values to ensure that connectivity is possible.

Metadata

The metadata support interface must be able to access and parse metadata
made available by each Grid middleware. Each middleware has its own format and schema for specifying metadata, hence parsers are required for each one to normalise the Execute Job node.

**Job and Data Service Management**

The job and data service endpoints for identifying interfaces used by each middleware must be supported and handled in order to invoke and monitor services, transfer data between the workflow system and Grid servers as well as handling data transfer between different Grid servers where applicable.

**4.4.5 Adaptive API Approach Interactions**

The interactions between the components based on the Adaptive API approach follow the same interactions during authoring and execution as those defined for the DTE-M node mapping approach. Figure 4.9 shows an overview of these Adaptive API based interactions.
Like the DTE node mappings, the upload and download data interact with the data service, in this case an abstract data service. Likewise, the execute job interacts with the abstract and data job services. Depending on the parameters and endpoints, the workflow contacts the correct data and job services for each middleware at runtime in order to invoke the service.

### 4.4.6 Adaptive API Approach Implementation

The adaptive API implementation uses a DTE-M node mapping approach. Like the Gateway approach, these implementations are based on the...
InforSense workflow system accessing GRIA and GOS services. Changes to the node definitions, in terms of parameters, provides support for accessing both forms of middleware.

The Adaptive API implementation of the Upload data node has dynamic parameters, which change to reflect the middleware selected. Selecting a GRIA data service endpoint displays the same parameters as the GRIA implementation. As GOS uses FTP and FTPS for data transfer, selecting a GOS server allows the workflow author to select an FTP site from an administration specified list. Similar to GRIA implementation, the administrator who is responsible for providing the security details, hence the administrator provides credentials for accessing the FTP site.

The Execute Service node also supports dynamic parameters, which change depending on the Grid (GRIA or GOS) job service endpoint selected, with the GRIA parameters remaining unchanged compared to the DTE-M mapping. Note that the current GOS implementation currently lacks metadata hence the workflow author must specify parameters, ports and service URI manually as described in the R-DTE mapping approach.

The Download Data node remains unchanged in terms of parameters, as the Data Service access information required to download results from each middleware is stored in the data model passed to the node.
The Adaptive API-based Grid handler implements an abstract version of the data reference model to support both GRIA and GOS data objects. The workflow passes between the Grid nodes and contains methods for storing and persisting data on each middleware. Connecting a GOS data reference object to a GRIA service node results in automatic data transfer to the GRIA server for use with a GRIA service and vice versa (if the servers can contact each other). A warning is automatically generated to notify the end user that this data.

4.4.7 BRIDGE Case Study: Updated DockFlow

The adaptive API approach can replace the Gateway approach applied to the DockFlow scenario. The same environment for the Gateway approach is used, with docking tools deployed across GRIA and GOS services depending on their location. The proxy server GRIA server is no longer required since the workflow supports an Adaptive API based Grid handler. Constructing the workflow branches for accessing the GRIA deployed docking tools (AutoDock and FlexX) remains the same as for the gateway approach. These tools are now directly accessed using a DTE-M based GRIA v5 handler based on the adaptive API. The Adaptive API handler accesses the GOS deployed docking tools (DOCK and GAsDOCK) directly rather than going through a proxy server. Constructing the workflow branches however is more difficult due to the lack of GOS metadata. The workflow author must know the details for GOS service to access in order to
specify the number of inputs/outputs and service URI for the Execute Service node.

Figure 4.10: Revised DockFlow using the Adaptive API approach

Figure 4.10 shows a revised version of the DockFlow prototype using the Adaptive API rather than the Gateway approach to access the Grid services. Further changes to the pre-processing steps moved certain high performance steps to Grid deployed implementations.

Case Study Evaluation

Whereas the gateway approach pushes the interoperability to the Grid level, the Adaptive API moves interoperability to the workflow level requiring more
development from a workflow perspective. By directly accessing both GRIA and GOS services, there is no need for a proxy server, thus overcoming some of the disadvantages of the Gateway approach. Removing the need for a proxy server reduces failure rate compared to the gateway approach and potentially improve workflow execution time as there is no need to duplicate and transfer data between two Grid servers and job submission calls/polling occurs directly.

The Adaptive API approach also has a number of limitations however. Firstly, utilising an Adaptive API requires the support of multiple methods to interact with each Grid middleware supported. This can complicate the node and handler implementations. Similarly, the node interfaces and administration tools themselves become more complex as specific parameters are required for each middleware supported. In the GRIA/GOS implementation, the lack of metadata had a great impact on the node design and usability of the Execute Service node. Naturally, similar Grid servers, in terms of structure and interfaces, work better in an Adaptive API environment where there are fewer compromises in terms of supported functionality and the parameters required at the workflow node level remain relatively close.

4.5 Effect of Service Locality on DockFlow Design

With a scenario like DockFlow, design decisions about where services run, how they are deployed and how they are accessed by a workflow system
Framework Extensions for Addressing Grid Heterogeneity

have a huge influence over the performance and usability of a workflow. This section investigates possible methods for deploying and accessing services as well as the affect these methods have at the workflow level.

4.5.1 Local and Remote Services

With an application such as DockFlow, all the docking tools are implemented as Grid services to take advantage of the high performance computing resources. Taking a similar approach for data pre-processing services is not necessarily advantageous and the properties of local and remote services must be taken into account to determine which combinations are the best implementation strategies.

Local services are located on the same machine as, or a machine accessible by, the workflow server. They are usually within an organizational or departmental network, securely behind a firewall to prevent unauthorized access and protect sensitive data. Local services have the advantage of reduced overheads compared to remote services, when performing operations such as data transfers.

The use of local services however, may require applications to be installed within an organization or department that may not have the required computational resources or environments to get the best out of the applications. Remote services such as Grid services overcome these
Framework Extensions for Addressing Grid Heterogeneity

problems by providing accessible applications on high performance compute machines. However, data must be sent and retrieved from a remote location, thus data transfer overheads can become an issue and in some cases negate the benefits gained from moving to remote service deployment. Given that the services are remote, security and trust must be set up between end users and service providers to ensure protection of data and insurance of results. Due to these disadvantages, it may not be feasible to deploy small applications as remote Grid services. Taking these issues in consideration is important when accessing the pre-processing operations from a workflow system to build an application such as DockFlow.

4.5.2 Implementation of DockFlow Pre-Processing Steps

The original implementations of the pre-processing steps for the docking tools were a combination of scripts and applications, each accessed differently from the InforSense workflow tool:

- Local Installation/Generic Components: For AutoDock, the workflow accesses the pre-processing scripts and applications using the general, local application integration framework supplied with the InforSense workflow system. These components are typically generic extensions to the workflow system and require customization from the client to support for each script and application. They also require local installations of a number of third party tools.
• **Local Installation/Bespoke Components:** Figure 4.11 shows an example of a set of specially developed nodes for accessing third party applications, in this case DOCK/GAsDOCK. Each node defined provides access to a particular locally executable application. However, local installations of the applications and their associated scripts are still required for this approach. Furthermore, parameters for each application are hard-coded in their associated node implementation.

![Figure 4.11: DOCK pre-processing steps using the local installation/bespoke components method](image)

• **Remote Installation/Generic Components:** Involves accessing applications remotely through Grid or web service nodes (such as those discussed earlier on). This approach was adopted for FlexX due to commercial licensing requirements.

Service location methods and their accessibility from workflow systems generally fall into the aforementioned categories.
4.6 Heterogeneity Performance Evaluation

The following evaluations compare the Gateway and Adaptive API based architectures in terms of performance, usability, the setup process and maintenance. A performance and usability based evaluation of the three service deployment methods used to implement the DockFlow pre-processing steps has also been conducted.

The environment for comparing the Gateway and Adaptive API approaches consists of DOCK and GAsDOCK applications deployed in the People's Republic of China using GOS Middleware. A GRIA gateway is also installed within the same institution to reduce bandwidth overheads and keep the experiments as unbiased as possible. A standalone InforSense workflow system in the UK accessed these services. For the pre-processing comparisons, the AutoDock and AutoGrid applications were executed at an academic institute in Egypt. To test local services as part of the preprocessing comparisons, AutoGrid was also installed locally on the same server as the InforSense system for access via general command line and custom bespoke nodes.

4.6.1 Comparing the Gateway and Adaptive API Approaches

Table 4.1 and Figure 4.12 show how data transfer performance changes as the data size is increased. In this set of tests, data was uploaded from and
then directly downloaded to the workflow machine. The overheads caused by the gateway approach transferring data between two Grid servers is significant when compared to the adaptive API approach which only requires a single data transfer. Both approaches have a workflow overhead, but the Gateway approach also has an additional overhead associated with the proxy Grid server invoking the data service for the second data server. Overall the difference between the overhead is linear in terms of the data size implying a constant overhead cost per data element transferred.

<table>
<thead>
<tr>
<th>Data Size (Mb)</th>
<th>Gateway (s)</th>
<th>Adaptive API (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.7</td>
<td>2.1</td>
</tr>
<tr>
<td>5</td>
<td>7.3</td>
<td>6.2</td>
</tr>
<tr>
<td>10</td>
<td>17.4</td>
<td>9.7</td>
</tr>
<tr>
<td>15</td>
<td>19.4</td>
<td>15.8</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of data transfer times of Gateway and Adaptive API approaches
Table 4.2 and Figure 4.13 compare the job execution times for the DOCK application. For each of these service invocations, the simplest test case involving the use of only a single molecule is measured. Workflow systems add a noticeable overhead compared to directly accessing the middleware irrespective of the heterogeneity approach used. The Gateway Approach however produces more overheads compared to the Adaptive API approach through the preparation, creation and submissions of the JSDL file associated with GOS service invocations.
Table 4.2: Job Execution times for direct GOS access, Adaptive API and Gateway approaches

<table>
<thead>
<tr>
<th>Execution Method</th>
<th>DOCK (s)</th>
<th>GAsDOCK (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOS Direct</td>
<td>97.4</td>
<td>67.4</td>
</tr>
<tr>
<td>Adaptive API</td>
<td>120.8</td>
<td>89.7</td>
</tr>
<tr>
<td>Gateway</td>
<td>297.4</td>
<td>234.6</td>
</tr>
</tbody>
</table>

Figure 4.13: Execution times for direct, Adaptive API and Gateway approaches

Table 4.3 and Figure 4.14 indicate that the service execution time varies as the number of molecules for each experiment increases. The overheads caused by the gateway approach are consistent, as the extra setup and invocation time generally remain the same regardless of how long the job takes to execute.
Table 4.3: Job execution times for the Gateway and Adaptive API approaches as the number of molecules used increases

<table>
<thead>
<tr>
<th>No. of Molecules</th>
<th>Gateway (s)</th>
<th>Adaptive API (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>234.6</td>
<td>89.7</td>
</tr>
<tr>
<td>5</td>
<td>273.5</td>
<td>124.8</td>
</tr>
<tr>
<td>10</td>
<td>306.3</td>
<td>157.8</td>
</tr>
<tr>
<td>50</td>
<td>541.6</td>
<td>386.4</td>
</tr>
<tr>
<td>100</td>
<td>727.8</td>
<td>616.5</td>
</tr>
</tbody>
</table>

Figure 4.14: Job execution times for the Gateway and Adaptive API approaches as the number of molecules used increases
4.6.2 Comparing Pre-Processing Operations

Two factors need to be considered when it comes to measuring the performance of service executions with respect to service deployment. The first factor is the time it takes to execute an application, while the second is the time it takes to execute the application and perform data transfer. Table 4.4 and Figure 4.15 show the results of an AutoGrid execution with respect to the first factor. AutoGrid is a pre-processing application used to produce and format the files used with AutoDock. Two installations of the AutoGrid were deployed. A local installation on the same machine as the InforSense was accessed directly via command line scripts to capture the native execution times. This installation was also accessed via the workflow system using the bespoke toolkit (custom, specifically designed nodes) and the generic command line integration nodes. The Grid deployment in Egypt (using GRIA) was accessed using the GRIA plug-in. Note that as expected, the remote Grid based installation running on high performance machines performs better compared to the local deployment. For local services, the bespoke integration performs marginally better than using generic nodes as it directly accesses the application.
<table>
<thead>
<tr>
<th>Execution Method</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>101.29</td>
</tr>
<tr>
<td>Bespoke</td>
<td>102.44</td>
</tr>
<tr>
<td>Generic</td>
<td>103.25</td>
</tr>
<tr>
<td>Grid</td>
<td>24.09</td>
</tr>
</tbody>
</table>

Table 4.4: AutoGrid (AutoDock pre-processing tool) execution times

![Graph showing execution times](image_url)

Figure 4.15: AutoGrid (AutoDock pre-processing tool) execution times

Table 4.5 and Figure 4.16 shows the performance of AutoGrid including data transfers required to get the data to the AutoDock application. Since AutoGrid is installed remotely within the context of the DockFlow scenario, the input data for AutoGrid needs to be transferred, whereas for the local installations the outputs from AutoGrid are sent to the remote server for use with AutoDock. While data transfer affects the performance in all instances,
the data transfer times for the local installations are greater as the outputs for AutoGrid are larger than its inputs.

<table>
<thead>
<tr>
<th>Execution Method</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bespoke</td>
<td>154.96</td>
</tr>
<tr>
<td>Generic</td>
<td>156.49</td>
</tr>
<tr>
<td>Grid</td>
<td>40.51</td>
</tr>
</tbody>
</table>

Table 4.5: AutoGrid execution times with data transfer

Figure 4.16: AutoGrid execution times with data transfer

4.7 Summary and Conclusions

This chapter described how the SOA based Grid workflow framework can be successfully extended to support two forms of interoperability, the Gateway
and Adaptive API approaches, as well as investigate how service deployment and access decisions affect the workflow composition process and performance. Both forms of interoperability are viable approaches for achieving Grid heterogeneity, however each methods comes with its own strengths and weaknesses.

With the Gateway approach, heterogeneity occurs at the Grid layer with a proxy server used to handle Grid interoperability. Responsibility therefore rests on the Grid middleware developers to develop the connectivity interfaces while maintaining a consistent level of usability. Few or no workflow changes need be made to the handler to support the Gateway approach, thus presenting a common interface to workflow authors and maintaining Grid flow usability. Performance wise however, the extra data transfer and invocation methods add increasing overheads to the overall workflow execution time, and increase the chance of failure.

With the adaptive API approach, interoperability moves to the interface layer. This move requires some development at the Grid level and major changes at the workflow handler level, but resolves the performance setbacks produced by the gateway approach.

The DockFlow scenario identifies a need for various service deployment and access methods, combining local and remote services for the Docking tool pre-processing steps. Remote services do away with local installations,
however data transfers between local and remote machines can add unnecessary overheads, and counteract the advantages of remote service deployments. Not all services are applicable for Grid deployment. Services, which do not require the security or data handling capabilities provided by Grid middleware, suffer from such overheads, and are better deployed as simple web services or local services. Bespoke nodes also have greater usability compared to more general service access nodes, as they are designed for a specific purpose and with the scientists in mind. However, building specialised nodes entails more development and maintenance when compared with generic nodes, and also prevents solutions from being built on the fly.
Chapter 5

Supporting High-Level Workflow Development

One of the aims in solidifying the two halves of the simple SOA model, with a reusable Grid-workflow framework in a heterogeneous environment, is to maintain a level of usability. By introducing Grid connectivity, however, workflow building becomes inherently complex. The goal is to hark back to the intuitive workflow construction of the single node mapping (one node per service or operation) while maintaining the benefits of the DTE-M based node mappings. Applying further abstractions to aid both service discovery and service composition is one method of addressing this problem. Such a solution fits the extended SOA models which introduces a third component that represents Discovery Agencies. This extra component aids or carries out the decision-making tasks that are performed by the workflow author. The extension also approximates workflow optimisation, since selecting the best services to meet user requirements can improve performance and the quality of results.

Node abstraction facilitates this automatic service selection, leading towards semantic service discovery and composition, and reducing the complexities associated with building workflow applications. An abstract workflow node is
not tied to a particular implementation of a service, and allows the workflow author to describe the task at a higher level. Abstract nodes must be mapped to a concrete implementation of a service in relation to the Grid-workflow framework. Automated runtime selection between services provides the basis for semantic service resolution. SIMDAT and BRIDGE project case studies and scenarios evaluate this SOA model abstraction extension.

### 5.1 Supporting Abstraction

Most scientific workflow systems as mentioned earlier, support both data and control flow constructs. However, due to the different modes of invocation and co-ordination, there is a distinct layer separation. The concept of “Grid Flow” is in line with this “flow layer” separation, referring to a data flow specific to performing Grid service co-ordination and execution. As discussed previously, it is necessary to keep the distinction between data and Grid flow when composing workflows to avoid incompatibilities. This distinction also has another purpose; it allows for the grouping and describing of Grid nodes at a higher level of abstraction. Curcin [25][26][28] models generic methods for abstracting data flows and embedding workflows within workflows. Most workflow systems support different flow level distinctions, allowing them to support this Grid flow methodology. InforSense has the concept of a group node that encapsulates nodes within a workflow and service nodes, which are workflows or sub-workflows with node ports and parameters promoted to
the workflow level allowing workflows to appear as nodes within other data flows.

The Pegasus [34] workflow system supports abstraction and has been used with systems such as DAGMan and Kepler. Pegasus is a fully functional system and a modularized set of components that allows for the interchanging of technology implementation. Furthermore, Pegasus is suited for abstracting scheduling based workflow systems rather than complex scientific workflows that have their own well-defined interfaces. Giannadakis [52] proposes and implements a workflow abstraction base on lambda lifting for the Discovery Net platform. Other workflow systems have similar forms of abstraction such as KNIME, which uses the concept of Metanodes for encapsulation. This chapter describes extensions to the abstraction work done by Curcin and Giannadakis, with respect to Grid services, and applying them to the Grid workflow layered architecture. The work presented here also investigates resolving abstract nodes to Grid nodes or workflows for execution.

By separating the Grid flow layer from the other layers, it is possible to start grouping the Grid service interactions to generate sub workflows or workflow fragments that consist of a combination of Upload data, Execute Job and Download data nodes. Workflows represent each of these fragments as a single step in a Data flow and forming the basis for workflow abstraction and eventually semantic composition.
Figure 5.1: Encapsulation of DTE-based Grid workflows

Figure 5.1 shows how Grid flows can be represented as steps in a data flow. The aim of using abstraction is to represent and describe these steps at high-level, to support mapping of abstract nodes to Grid services and Grid flow fragments. Achieving this abstraction is possible with abstract nodes, which form abstract workflows when combined. The following definitions describe the support for abstraction in the Grid service selection and composition processes:

- **Concrete Workflow**: A workflow that is complete and well-defined for submission to the workflow engine for execution. All the nodes used in a concrete workflow map directly to known, existing services with well-defined properties.
- **Abstract Node:** A node in a workflow that is not bound to any particular service, in our case a Grid service. Abstract nodes simply act as stubs or placeholders in the workflow during workflow authoring and need to be resolved before or during execution.

- **Abstract Workflow:** A workflow defined in terms of abstract nodes.

Note that workflow systems are generally not able to execute abstract nodes in the traditional sense. Before execution, workflow systems are required to map abstract nodes to concrete implementations consisting of one of more Grid services. This mapping is either a manual or an automatic process (in the case of automatic, conducted by an intelligent system).

![Diagram](Abstract Workflow)

*Figure 5.2: Mapping a three node workflow to particular implementations of available services*
Figure 5.2 shows a simple chain of three abstract nodes, with each node performing a specific function. Initially, none of the nodes in the workflow are mapped to a concrete service. To move from an abstract workflow to a concrete workflow, each node needs to be instantiated or associated with a concrete, Grid service based implementation.

### 5.1.1 Mapping Abstract Workflows to Concrete Workflows

Resolving abstract workflows in order to generate concrete workflows is an iterative process. The first step is to define an abstract workflow using abstract nodes to represent key tasks. At the most basic level an abstract node requires input and output ports allowing the node to connect to other nodes and propagate metadata through the workflow. It is possible to add further definitions such as port types and names, parameters (with types and names) and keywords to the abstract node to perform more sophisticated service searches and resolution.

After the definition of an abstract workflow, applicable services to resolve each task must be found and located. After discovering an existing service, which could potentially meet the requirements of the desired task, the properties of the service require checks in order to identify how well it meets the requirements.
In some cases a single service may not be sufficient to address a task, therefore a combination of services are required. Upon identifying the services to use in the workflow, connections formed between each service ensure the flow of information. Ensuring this flow of information may require modifying the workflow by adding additional nodes to bridge the gap between disconnected nodes.

Note that supporting a mix of abstract and data flows, where some nodes are abstract and other nodes are associated with a specific implementation, provides a key advantage by allowing workflow authors to refine their workflows.

5.1.2 Service Mapping Categories

Mapping from abstract nodes to concrete services can vary depending on the context, descriptions applied to the abstract node and the services available. Abstract-to-concrete service mappings can fall into one of four categories. Concrete Services can either offer the full, required functionality, i.e. functionally equivalent, or similar functionality. Likewise, the interfaces can be either exact or different:

- Functionally equivalent with exact interfaces (FE-EI)

- Functionally equivalent with different interfaces (FE-DI)
• Functionally similar with exact interfaces (FS-EI)

• Functionally similar with different interfaces (FS-DI)

The four service mapping categories use at least one of the following three properties for determining the ranking for service selection, service interfaces, QoS constraints and service fidelity:

• **Service Interfaces:** can vary in terms of number of interfaces and the types of data accepted. This property is important for determining the flow of data and connectivity within a workflow.

• **QoS Constraints:** are properties that govern the execution of a service such as time, cost and resources used.

• **Service Fidelity:** how closely the service matches the initial requirement in terms of function.

Table 5.1 shows which of the three properties applies to each of the four service mapping categories.
The importance of each of the properties varies depending on the method of service selection (as shown later in this chapter) and user requirements, i.e. if the user’s aim is to minimize costs as much as possible then QoS constraints, specifically time, take precedence. We will look at the application of these parameters later on in the next chapter.

**Functionally Equivalent with Exact Interfaces**

FE-EI services are services that perform exactly the same function and have the same interfaces as specified in the abstract node, but can still vary in terms of QoS constraints. For example, consider two deployment of the same application on different machines. The first deployment runs on a dedicated high-performance compute machine whereas the second runs on a standard desktop machine. The first machine would execute the service quicker, but the monetary cost may be higher due to the resources. In such a case, the selection of the service depends on the author’s main criteria – time or cost.

---

### Table 5.1: Properties associated with service mapping categories

<table>
<thead>
<tr>
<th></th>
<th>FE-EI</th>
<th>FE-DI</th>
<th>FS-EI</th>
<th>FS-DI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service Interfaces</strong></td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td><strong>QoS Constraints</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Service Fidelity</strong></td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

---

197
**Functionally Equivalent with Different Interfaces**

FE-DI services are services that perform exactly the same function but may vary in terms of inputs, outputs and parameters, i.e. in terms of names or structure of inputs/outputs. QoS constraints are also applicable and therefore the service selection becomes more complex as there are now two sets of variables for determining the selection. Consider an example of a workflow for performing image processing. Two services may be available for performing the same required function. The first implementation takes a JPEG image, while the second takes a PNG. The choice of service to use depends on the format of the file going to or coming back from the service. Likewise, a one deployment of docking tool may require the protein sequence and a parameters file, while another takes both files in an archive. Service interfaces that do not match with the workflow inputs and outputs can break workflows and later, this chapter describes how this affects the service selection process. To resolve interface mismatches using auxiliary tools known as shims (discussed in detail in the next chapter) is one possible method for bridging the gaps between nodes. These tools can range from simple data formatting to performing operations to produce the correct data. The next chapter also discusses models associated with functionally equal or similar services with different interfaces.

**Functionally Similar with Exact Interfaces**

FS-EI services are services that have exactly the same interfaces but vary in the details of their functionality. For example, taking the protein docking
scenario, an abstract node specifying docking as its functionality, requiring a ligand and receptor as an input can be mapped to either a GAsDOCK or DOCK service which are both based on the same platform but have different methods for performing protein docking. Both take the same inputs and produce exactly the same results structurally, but the actual data can vary. In situations like this, a particular tool can have a higher ranking as it better matches the user requirements. If the requirements are more relaxed, i.e. either docking tool meets the requirements, or defined at a coarser granularity, then it is possible to use QoS parameters alone for choosing between services. Language such as WSMO and OWL-S can be used to define the functionality of a service and used for service comparison.

**Functionally Similar with Different Interfaces**

FS-DI services perform similar tasks and vary in terms of their interfaces. For example, consider an image-processing scenario. An abstract node represents the task of applying a radial blur on an image. The image required and expected back is a JPEG object. The closest matching service performs a Gaussian blur and returns a GIF object. Like FE-DI services, FS-DI services may require data transformations to bridge the gaps between service invocations. Note that selecting FS services may have implications further along in the workflow and may require modifications and requirements to parts of the workflow or the whole workflow. FS-DI services are the most difficult to resolve as all three properties can be taken into consideration when performing selection.
5.2 Abstraction-Based Grid-Workflow Architecture

In order to support abstraction in a workflow, Grid workflow architectures require an Abstract Handler. Essentially, it is possible to build this support directly within the Grid Handler, but the separation allows Abstract nodes to be resolved to any kind of service (Grid, web or variations of local services) depending on the mapping. Adopting a DTE based node mapping approach requires resolving abstract nodes to fragments of Grid flow (i.e. a combination of upload, execute and download nodes); hence a registry of available Grid flow fragments is required along with names, parameters and port data for service discovery. Each fragment consists of a combination of upload, execute and download nodes, which are required to perform a service invocation. To support abstract resolution, a domain expert describes and publishes each entry in the registries for discovery by the workflow system. This initial design performs simple name matching to identify and refine the potential list of services. The workflow author then picks the most suitable service and the returned Grid flow fragment replaces the abstract node.
Figure 5.3: Extended layered architecture to support abstraction

Figure 5.3 shows a revised layered architecture that supports the construction of abstract workflows that resolve to Grid flows. Note that the Grid Handler must also be deployed in order to render the workflow fragments returned otherwise the nodes will be recognised, breaking the workflow.

5.2.1 Abstract Handler Definition

Figure 5.4 below shows the definition for the Abstract Handler, showing the layers and components required to add support for abstraction. Note that the layers in the Abstract Handler are similar to the Grid Handler.
An abstract handler consists of the following layers:

- **Node Definition**: A single node that can be configured to reflect the abstract descriptions, and chained together to make an abstract workflow.

- **Workflow Based Support Interfaces**: Interfaces to support abstract workflow creation that include adding annotation to nodes and workflows, submission of an abstract workflow to a resolver and the retrieval of the concrete workflow specified in the workflow language and rendered in the workflow client.

- **Workflow Resolution Interfaces**: Consists of interfaces used to resolve the abstract workflow and return the concrete workflow.
5.2.2 Defining the Abstract Node

The abstract node is a placeholder node with no parameters and only one output port by default as all workflow nodes must return one output. The node has parameters for entering the number of extra input and output ports. The number of ports associated with this node dynamically change with the entered number. Abstract nodes must satisfy type constraints and enable connections to other nodes. Therefore, by default, all inputs and outputs handle file reference objects, but are modifiable to support other types. The node invokes the abstract submission, contacting a registry and sending information, which includes the name and number of ports. The workflow author must be able to view the results from the registry, which is possible through a GUI that displays the URI, description and input, output details for each service in order of best matches. Upon service selection, another service call retrieves the workflow fragment, which replaces the abstract node. Note that abstract nodes should not be executable and prevent a workflow from executing until each abstract node has been resolved to a concrete implementation. By matching the port types (and preferably port names) the node resolution should not break any pre-defined any workflow connections.
5.2.3 Abstract Support Interfaces

The following support interfaces are required for abstract component support. Note that the separation between these interfaces and the abstract node allows for changes to the support interfaces without affecting the abstract node definition.

Abstract Node Submission

This interface handles the submission of the workflow representation with annotations to resolution tools. Like with typical workflow execution this occurs through the workflow server.

Service Retrieval

The service retrieval interface is responsible for retrieving the resolved abstract workflow and rendering it the workflow client to allow the workflow author to view and modify the workflow before execution. Authoring time checks performed on the returned workflow highlight any potential issues, such as missing compulsory parameter values, or unresolved node connections.

Registry Interaction

A custom interface is required to interact with the registry and allow the workflow author to perform service selection. The interface takes the values returned by the query and displays them to the workflow author at authoring
time. The user can browse the matching services and perform a selection. After the selection of a service, the interface submits the selection to the concrete workflow retrieval interface, which downloads the XML workflow fragment.

5.2.4 Registry

The information used to populate a registry can vary greatly and affect the discovery and composition process. For example, the simplest form of discovery would involve retrieving an endpoint and place it in a Grid node skeleton. However, using this form of composition would require further service calls to retrieve metadata to determine ports and composition to add data upload and download nodes. The method adopted here requires housing workflow fragments in the registry, which are essentially XML statements readable by appropriate workflow clients. The advantage of this method is that the full range of Grid operations are returned with each service; upload data, execute job, data and resource allocation depending on the node mapping used or Grid middleware involved. One disadvantage of storing workflow fragments is that they require authoring and then submission to registries, which can be a time consuming process. Secondly, producing fragments ties the registry to particular workflow implementations. This problem can be overcome by storing abstract workflow information (endpoints, number of inputs, outputs, etc) and using an intermediate
Supporting High-Level Workflow Development

brokering system that takes this abstract information and returns workflow fragments in XML.

For each fragment a name must be provided for discovery and relate to the function of the service invoked. Along with the name, the registry must also provide for each service a service URI, metadata and number of inputs and outputs. This extra information is required for performing matches that are more accurate, and form the basis for the registry design for semantic service discovery. The registry must provide a download link in order for the workflow to obtain the workflow fragment. This fragment then replaces the abstract node in the workflow and rendered in the client.

Note that the registry requires an API or accessible interfaces (such as web services) in order for a workflow system to make the service calls to retrieve information from the registry, perform the search and download the workflow fragments.

### 5.2.5 Interactions

Figure 5.5 shows the interactions for submitting and resolving an abstract workflow.
Upon execution, the workflow submits each defined abstract node to the abstract workflow submission interface. This interface extracts the terms required for service matching and submits them to the registry. The handler returns and displays the services that match the terms. Once the workflow author selects a service, the workflow retrieval interface downloads the workflow fragment and processes the abstract node to replacing the abstract node description with the workflow fragment. The workflow client is refreshed to render the changes. Workflow authors can then edit workflow fragments like any other node in the workflow.
5.2.6 Implementation for Supporting Abstraction

In order to support the abstract node a set of workflow fragments accessing GRIA services are required. Information about these fragments, and the services they invoke, need entering into the registry.

Workflow Fragments
These fragments use GRIA nodes and deployed by InforSense as service nodes. This involves promoting the parameter associated with the GRIA service arguments, the input ports to any upload data nodes so that it is possible to pass a file reference along, and the output port of one or more download data nodes for execution. At the workflow language level, each workflow fragment has its own DMML representation. The ports and parameters of a service node mirror the promoted ports and parameters of the underlying workflow fragment. At the DMML, the service node references the path to DMML for the workflow fragment.

Registry
To simplify the integration process, rather than using a third party registry, this implementation uses the InforSense Userspace to host workflow fragments. The support classes in the Abstract handler use the InforSense API to access the Userspace, iterate through the available service nodes, extracting the names, descriptions and input/output from the DMML.
Metadata

In order to map to the abstract nodes, the following information is required in the metadata associated with each workflow fragment. This metadata is included within the DMML and is hence in XML format. Each input and output includes a reference name and type of object. Each deployed parameter includes a reference name, type and default values.

Abstract Node

Like the GRIA nodes, the abstract node is developed using the InforSense API. The node consists of three parameters; the first to specify the number of inputs, the second to specify the number of outputs and the third parameter for submitting then abstract node for resolution. Each input and output added accepts any object for input and returns output metadata for any object. It is not possible to invoke the node and it has no process method associated with it. Submitting the abstract node extracts the name and number of inputs/outputs from the DMML and sends the values to the registry. The registry looks for the name of the node in the service URI, description and compares the number of inputs and outputs. The node displays the best matching services at the top of the list. If no service matches the abstract node, the user has an option to see all available services, or refine the abstract node. Figure 5.6 shows the results from the registry displayed in a pop-up GUI associated with the node. Selecting a service makes another call to retrieve the chosen workflow fragment. The group node containing the encapsulated workflow fragment replaces the abstract node and the client is
refreshed. Note that in InforSense service nodes are expandable allowing workflow authors to view and edit the inner workflow.

Figure 5.6: Abstract workflow interface for retrieving matching workflow fragments

Where input and output ports match to the service returned (based on the name and types in the metadata), the connections, InforSense retains the connections between the service node and the other workflow nodes. However, if there are node port mismatches then the connections between nodes are broken, requiring re-establishment in order to produce an executable workflow. In cases where the workflows break, workflow shimming may be required using intermediate nodes or services, as covered in the next chapter.
5.2.7 Case Study: Abstracting DockFlow

The following case study extends the Protein Docking application scenario described in the previous chapter by introducing abstraction, allowing for the selection of docking tools deployed at different sites. The aim of the scenario is not to test performance, but the validity of the support for abstract workflows and whether the framework produces expected results.

Abstract DockFlow Scenario

Testing the abstract handler developed for the InforSense workflow uses the same technologies as the DockFlow scenario previously described. This particular scenario focuses on the AutoDock invocation and processing branch. In this scenario, two AutoDock services are deployed across GRIA servers at a research institute in Germany and an academic institute in Egypt. The service deployed in Germany takes three inputs, a receptor file, ligand file and grid parameter file, and returns the docking results. The service deployed in Egypt takes an archive containing ligand and receptor files. This implementation also invokes both the AutoGrid and AutoDock services and returns the AutoDock archives. Clients can access these services via workflow fragments, published in the InforSense Userspace for discovery.

The workflow author specifies an abstract node that takes two inputs, the ligand and receptor files from the database, and an output for the docking
results, as well as applying a label “AutoDock”. Submitting the abstract for resolution submits the name and number of ports to the registry. Based on the name, the workflow returns both implementations of the AutoDock service. Figure 5.7 shows the returned workflow fragment added to an AutoDock workflow. Selecting the Egypt implementation requires the need for a shimming tool to archive the ligand and receptor for submission to the GRIA service. The workflow author can perform this shimming process manually. Selecting the Germany implementation requires an AutoGrid service to produce the grid parameters file required for the AutoDock service. It is possible to add an AutoGrid service to the workflow if the AutoDock pre-processing applications are available locally, relative to the workflow server.

Figure 5.7: AutoDock workflow fragment requiring manual shimming
Alternatively, placing an abstract, node that resolves to an AutoGrid service, between the ligand/receptor files and the AutoDock workflow fragment, can potentially overcome this problem. Searching through the registry returns an AutoGrid deployment available on the Germany server. Selecting this service returns the appropriate workflow fragment, which bridges the gap and produces a complete workflow. Note that by combining both the AutoGrid and AutoDock service a single workflow fragment can provide access to both services in the logical order.

Keeping with the two inputs and one output abstract node definition, the workflow author defines a new node with the label “Dock”. Providing a more broad term yields more matches in the registry. Along with the AutoDock service, DOCK and GAsDOCK workflow fragments are also available and satisfy the abstract node constraints allowing the workflow author to choose between implementations. Note that FlexX is excluded from the search results as it does not contain the term “dock” in its name or service endpoint. In such a case, the registry description must ensure that a range of valid terms is included in order for the workflow fragment to be recognized.

**Scenario Evaluation**

The scenario demonstrated an Abstract handler and associated components (registry and workflow fragments) for performing authoring time service selection by successfully resolving the high level descriptions for the protein docking scenario. The scenario also demonstrated how the decoupling of
Grid and data flows helps to support abstraction by making each Grid flow fragment viewable as a single data flow node. Note that this approach is heavily dependent on the accuracy and quality of the information stored in the registry, the abstract node definition by the workflow author and the production of workflow fragments that wrap the service. Inaccurate or lack of relevant terms in a description can lead to false positive matches or undiscovered workflow fragments. Using semantic ontological terms and languages that support them such as WSMO and OWL-S can help to normalize terms to prevent the aforementioned problems from occurring and help in providing higher-level descriptions, i.e. the workflow author may not know exactly what service they are looking for and can therefore specify key functions.

The current method focuses only on service selection and thus, for abstract workflow support, does not consider composition. Manual composition is required and in cases where services do not connect, manual shimming is required. Each abstract node can only be resolved to a single workflow fragment when in some cases a combination of fragments or services may be required in order to achieve the desired function, in which case the abstract nodes definition must be at a lower level in order to find the required services.

This form of service selection does not optimize the workflow, there is no way for determining for example, which implementation of AutoDock will
provide results the quickest. Again, semantics are required to describe the desired conditions, which govern the execution of a service.

5.3 Runtime Service Selection

So far, the focus has been on manual selection during workflow authoring time. Selecting services has a major disadvantage in that it is possible for the chosen service, or factors that govern the service execution, to change by the time it comes to execution. These changes can include changing the version of the underlying application used which affect the interfaces used to access the service, modifications to the access interface, network traffic, Grid server failure and multiple invocations of the same service at the same time.

It is possible to overcome the aforementioned disadvantages for authoring time service resolution by performing service resolution at runtime. However, the problem with runtime selection is that it has the potential to break node connectivity since the service best matching the requirements may not match the interfaces described in the abstract node. Therefore, it is only possible to perform runtime selection with either FE-EI or FS-EI services, in other words services with exact interfaces, but can vary in terms of functionality and QoS constraints.

Selecting between EI services either occurs randomly or based on a set of metrics to determine rankings. These metrics relate to the function of the
service, with the scoring based on how closely each service matches the abstract requirements. Metrics can also be used to select between services based on the quality of service, i.e. monetary or computational costs, execution time, resource available, etc.

**Knowledge Base**

Applying a knowledge base to service selection addresses the normalization and searching issues identified in the previous case study. Ontologies provide a set of terms for annotating and searching services as well as highlighting particular features of each service allowing for service ranking. Service ranking helps to distinguish between two equivalent services in terms of QoS or fidelity. For example, distinguishing between two implementations of the same application deployed on two different Grid services, each with varying performance metrics. Although they produce the same output, their performance can affect their ranking and determine the chosen service. Another example would be two services that perform the same function but one is more up-to-date than the other and would therefore take precedence. The disadvantages in applying a knowledge base include increased overheads due to more complex service-matching and increased preparation time in adding, updating and managing the annotations applied to services. Part of this chapter investigates the influence these disadvantages have when supporting runtime selection.
Semantic Broker

With the addition of a knowledge base, a semantically enabled brokering system is required to perform the service discovery and selection process on behalf of the end user, especially since this selection occurs during workflow runtime rather than authoring and is thus an automated process. The broker must have a set of interfaces to allow workflows, or indeed any third party applications, to access its functionality.

5.4 Extending Abstraction to Support Runtime Selection

With runtime selection, a tight coupling is required between the Grid nodes and the semantic interfaces; therefore, the Grid handler provides support for a semantic broker within the Grid handler as shown in Figure 5.8.
Figure 5.8: Layered architecture for automated runtime service selection

With this form of service resolution, a complete Grid flow requires an extra node for producing the job service endpoint, data service endpoint and service URI at runtime. The workflow propagates these endpoints to the Upload and Execute Job nodes at runtime overriding and endpoints pre-configured in the node. Depending on the choice of implementation and functionality needed, the Semantic Handler may be dependent on the Grid...
Supporting High-Level Workflow Development

Handler and required to perform service calls to the Grid middleware, such as performing validity tests to ensure that a job service, data service or application are reachable before submitting the values to the Grid nodes.

5.4.1 Grid and Semantic Broker Handler

Figure 5.9 shows the architecture for an updated Grid handler required for supporting runtime equivalent service selection.

![Grid Handler Diagram]

Note that this particular design for the Semantic Broker handler supports any of the node mappings defined earlier, however a resource allocation based mapping would also require the selection and submission of allocation parameters to the Grid nodes. A service selection node interfaces the semantic broker for querying and retrieving the endpoints to feed to the other
nodes. The model handles the transfer of this information to the subsequent Grid nodes. The service selection node uses the Broker Interface to interact with the broker.

### 5.4.2 Nodes

In order to use the Semantic Broker with the Grid nodes, changes to the Upload Data and Execute Job nodes in the Grid Handler are required. The described design uses the DTE node definition for the Grid Handler. This section highlights the handler component changes that are required to support a semantic-based architecture.

With DTE-based node mappings, the execute job node is configured at authoring time to determine the endpoint, service, ports and parameters. Upload and download nodes connect to this node forming a complete Grid flow. This same process is applicable when supporting service equivalence. However, rather than selecting a single endpoint multiple endpoints need to be selected. Therefore, changes to the upload and execute job nodes are required to support random selection from a list of endpoints.

The broker must return the following values in order to perform service equivalence selection from workflow systems:

- **Job Service Endpoint**: The job service endpoints for each service selected establish communication with the hosting Grid server.
- **Data Service Endpoint:** The workflow must transfer data to the correct Grid server before invoking a service, therefore the data service endpoint must correspond to the job service endpoint.

- **Service URI:** Depending on the deployment of the service, equivalent services can have different service URIs. The URI of the required services must also be provided and correspond to the job and data service endpoints.

As the job service endpoint, data service endpoint and service URI for each service need to be synchronised, a single interface is required to perform the parameterisation and selection of values.

**Service Selection Node**

The job and data service endpoints need to be propagated to the execute job node and each upload node so that when a service is selected at run time the values are sent to the nodes and used when the node is invoked. Like the execute job node, when a job service endpoint is selected a call is made to the Grid server to retrieve a list of services and the service URI can be selected.

**Service Selection Object**

When a service is selected automatically at runtime, the endpoints and URI for that service need to be submitted to the execute job and upload data nodes. A service selection object propagates these values.
Execute Job and Data Upload (Grid Handler)

An extra input port that takes the service selection model needs to be added to each upload and execute job node. The model must then be parsed with the service URI and job service endpoint accessed by the execute job node and data service endpoint accessed by the upload data node. Submission of these values to the Grid server is in accordance with and using the same interactions as the DTE-M mapping.

5.4.3 Semantic Broker Interface

The semantic broker interface is built on the Semantic Broker API and is responsible for communicating with the Grid handler to check validity of endpoints and is used by the Service Selection node. This support interface provides the GUI required for interaction with the registry, namely browsing ontologies to select the appropriate application. Note that this functionality could be directly implemented in the service selection node. However, by providing this separation, changes can be made to the selection process or to the Semantic Broker interfaces without affecting the node design and implementation. When a service is selected from the broker this interface tests the endpoints via the Grid Handler Job and Data services. If they are available the values are added to the model which is then passed to the Grid nodes, otherwise another service is selected and the process is repeated.
5.4.4 Runtime Service Selection (Broker) Interactions

Figure 5.10 shows the interactions for selecting a service at runtime from the registry.

The execution of the service selection node is synchronised with the Grid flow, i.e. when the workflow reaches the start of the Grid flow. The handler submits the selection term to the registry, which returns all matching queries. The registry interface selects the highest-ranking service and tests the endpoints to ensure the workflow can contact the Grid serve. If contact is unsuccessful, the node selects the next set of endpoints and repeats the
process. The handler adds the first set of successful endpoints to an instance of the service selection model. This model is submitted to both the Upload Data and Execute Job nodes where the relevant endpoint is extracted and used for the node invocation. Note that the execute job node must be parameterised at authoring time (i.e. utilizing metadata to define the ports) using one of the relevant services.

5.4.5 Runtime Service Implementation

The following components are modified or implemented to support runtime service selection. The Grid handler is this case is based on the GRIA middleware utilizing the DTE mapping.

The implementation for runtime service selection includes development of the knowledge base, semantic broker, semantic broker handler and extensions to the Grid handler. For this particular implementation the NEC semantic broker and ontologies used in the SIMDAT project are used based on the work done by NEC, Ontoprise, ULB and Fraunhofer IAIS.

Knowledge Base

The knowledge base includes ontologies and a knowledge model that connects multiple ontologies together. Ontologies allow for the setup of complete models making it possible to describe complex structures and encapsulate them at different levels of abstraction rather than the simple
attachment of key words to services. Once the ontologies, characterization of services and support models have been set up, each service instance needs to be annotated. This particular implementation uses an XML template for outlining the annotations and is linked to a service endpoint. To produce these XML annotations, the TUAM application was produced as part of the project.

The service ontology for this implementation is based on OWL-S which was extended to include specific constructs applicable to the type of services used in the SIMDAT Pharma scenarios. OWL-S was chosen as it describes fine grained services, where services have set inputs, outputs and parameters. The OWL-S extensions were added to support QoS constraints which can be applied to services.

**Semantic Broker**

The Semantic Broker houses the knowledge base (Ontologies, knowledge models and service annotations) required for the service discovery and a reasoner for semantic matchmaking. Annotations are published in the broker and rules are produced by a domain expert. The Semantic Broker also provides an advanced ranking functionality to select between equivalent services, e.g. selecting the service that has the most up-to-date database. Using a client API based on Web Services, a user can browse the ontology and a select a term representing the service required. At runtime this selection is passed to the reasoner which selects the best service based on
the annotations and rankings. The service endpoints are then returned back to the user. Note that the Semantic Broker is a dynamic environment with annotations being constantly updated, thus making it suitable for runtime service selection where the selection happens close to the execution. Figure 5.11 shows the architecture for the NEC semantic broker adapted from [91].

![Architecture of NEC Semantic Broker](image)

**Figure 5.11: Architecture of NEC Semantic Broker**

Although the web services can be accessed on the fly using workflow web service components, we prefer to build node based on the API to allow for more complex interactions with the broker, i.e. providing better user interfaces, such as tree viewers for browsing ontologies.
Upload Data and Execute Job Nodes

Each of these nodes is modified to accept an additional input, the service selection model. The process methods for the upload data and Execute job nodes are extended to extract the data and job service endpoints respectively at runtime and use these values to govern the execution of the node. Note that if a service is chosen which does not match the node’s the service fails and goes to the next service in the list.

Service Selection Node and Model

The service selection node is built upon classes generated from the Semantic Broker WSDL using the WSDL2Java tool provided by Axis. This node has a single parameter which launches a GUI for browsing the ontologies stored in the Semantic Broker knowledge base. As these ontologies can be large in size, only part of the ontology is returned. Every time a node is expanded, a web service call (SOAP) is made to retrieve the corresponding branch, which is then cached. As well as allowing the user to select a term, the GUI also allows the user to select the secondary rank from a list for use in cases where equivalent services are returned. The node has a single output which returns the service selection model. When the node is executed, the selected term and ranking criteria is submitted to the semantic broker. These results are submitted back to the node, cached in the node and a new service selection model is generated with the chosen endpoints. The workflow then submits the model to the Upload Data and Execute Job nodes as shown in Figure 5.12 below.
Before sending the endpoints to the Grid nodes, the Service Selection node tests the first set of endpoints to check if they are active or can be reached. If the node fails to do so, the next ranked service is tested. This process is repeated until a service can be reached or no more services are available in which case the node fails.

5.4.6 SIMDAT Case Study: IX-Odus Workflow

The layers separation ideology and runtime service selection enhancements were tested using a IX-Odus workflow defined by the pharma work package in the SIMDAT project. The aim of the IX-Odus workflow is to identify and annotate CDNA sequences coming from an in vivo expression experiment, namely the study of the Ixodes ricinus host-parasite interaction. The sequences in large number, coming from the DNA sequenator, are entered as batches and submitted to comparison and annotation steps controlled by a workflow design and enactor system. The analysis steps are dispatched to a series of remote servers where identification of similarities or relevant
structural or functional motifs are entered in a project database for further reference. This model is intended to serve as a case study for similar, possibly more ambitious studies of batches of unknown sequences, where more, and more diverse, bioinformatics servers and methods could be put in action.

**IX-Odus Environment**

This scenario consists of a number of bioinformatics applications deployed as GRIA services (BLASTX, BLASTN, InterProscan and Getorf) using GRIA v5. This particular scenario focuses on selecting between BLASTN services, thus various implementations of BLASTN are deployed at three locations; NEC, EMBL and ULB. The NEC semantic broker is deployed in Belgium to select between implementations. An InforSense v5 installation with a Semantic Handler (based on GRIA v4 and the DTE-M mapping) is deployed in the UK. This installation is used to create a workflow for the execution of the services.

**IX-Odus Scenario**

With the IX-Odus workflow new sequences are entered into the workflow by the user. These sequences are compared with existing sequences to check that they are indeed novel and not redundant. This is achieved by running a BLASTN application, deployed as a GRIA service, which compares the entered sequences to those in the BLAST-formatted databank. Any irrelevant or pre-existing sequences are annotated as redundant. If a sequence is
considered relevant for further analysis it is submitted to another workflow where the sequence is compared with further databanks and annotated by further applications (some deployed as Grid services others as local services) to identify functional and structural elements of the given sequence. If a hit is made at any of the steps, the sequence is added to the database. Otherwise, the sequence is passed to the next step.

For such a workflow the layer separation is paramount. A control flow is used to orchestrate each of the data flows that run analysis steps. In some cases the applications are deployed on the Grid requiring the Grid flow separation as shown in Figure 5.13.
The architecture for automated service selection is tested using the BLASTN operation within the IXODUS workflow. At runtime the node is to select between implementations of the BLASTN service running on different GRIA server deployments. It is necessary for BLASTN to be running and active at all times in order to continue the sequence analysis process and the preferred implementation is the one with the most up to date database. The registry is populated with BLASTN entries running at different locations and is updated as the databases are updated. To test the fail over functionality, certain servers were switched off to check if the next service in the list was chosen.

For the IX-Odus workflow a Grid Flow fragment (co-ordinated by a control flow) is built using one of the service endpoints. This is to determine the inputs and outputs beforehand in order to build the workflow. The Service selection node is then introduced and connected to the Grid nodes. Running a number of tests the Grid server we expect to be chosen is chosen. The preferred service also changes in accordance to the annotations in the Semantic Broker.

For further tests, the broker was populated with more entries, ontological terms and complex relations in order to test the performance of the workflow and affect on the overall execution to determine whether runtime service selection is a viable option.
Case Study Evaluation

The layer separation was successfully deployed in the ULB scenario and helped in both the authoring as well as troubleshooting process and workflow updates. For example it was easier to convert the Grid flows to runtime selection Grid flows as the layer separation helped identify where the Grid flows occur and each Grid flow could be modified without affecting the rest of the workflows. Likewise, using the control flow, improved the flexibility of the workflow allowing further analysis and database comparison tools to be added efficiently.

The main benefit of runtime service selection is that the chosen service is chosen close to the actual execution time and thus is the most accurate result. The fail over mechanism helps to maintain reliability by keeping the workflow active in case of service failure. The highest ranking service was returned in the tests conducted, unless the service was down in which case the workflow successfully switched to the next service in the list. A major setback with runtime service selection is that selection can only occur between services with equivalent interfaces. Like the abstract node, the accuracy of the selection is dependent on the terms entered and the quality of the entries in the Semantic Broker. For example selecting BLAST as a term is too high level to distinguish between different BLAST applications and could potentially yield BLASTN and BLASTX results leading to inaccurate matches.
Using semantic annotations we notice that as the ontologies increase in terms of size and complexity the greater the overheads. The first noticeable affect was the increase in authoring time due to the fetching of annotations from the broker. The more complex the ontology the longer it takes to browse for specific terms. We try to address this in the implementation through caching and loading only parts of the ontology rather than the whole tree in one go. Secondly, during workflow execution, the reasoning process also increased in time. While this overhead was noticeable, it does not have a huge affect on the overall workflow execution time and given the benefits, the tradeoffs can be justifiable in cases where execution time is not of utter most importance.

[91] details the decision making processes that occur in the semantic broker and highlights the performance of the queries in relation to the number of services annotated in the Semantic Broker.

The experiments in [91] show how the query time increases with the number of services supported by the broker. Querying over more features naturally has a greater overhead as the decision making process becomes more complex.
5.5 Summary and Conclusions

Abstractions can be applied to workflow systems and resolved to concrete Grid flows at authoring or runtime. Designs for abstract and semantic handlers that work in tandem with the Grid handlers, the Grid-workflow frameworks described in the previous chapter, and the discovery agencies (registries, semantic broker) address the extended SOA model. These approaches are flexible and modularized, allowing specific implementations of technologies to be interchanged. Different forms of service matching with respect to interfaces, functionality and QoS constraints have also been identified.

Authoring time selection is more intuitive as it allows for refinements with respect to service choices and requirements before a Grid workflow is executed. The authoring time selection process is a manually conducted affair with the workflow author making decisions about services and composition. The runtime selection process conducted in this chapter is an automated process and requires the use of semantic technologies for resolving services. However, as this form of selection is performed once the workflow is complete, the interfaces between the services must match, be known beforehand and based on services with interface equivalence.

The work in this chapter has only touched upon Grid service resolution. Ideally, an automated service selection process that allows for workflow
author interaction is required. The selection processes described focus on individual services, however for more realistic and optimized service selections, the entire workflow needs to be considered to put services into context. The next chapter looks at using intelligent agents for resolving abstract workflows automatically and focuses on the information required in order for them to make accurate decisions.
Chapter 6
Moving Towards Automated Service Composition Through the Use of Intelligent Agents

This chapter expands on the abstraction introduced earlier and identifies how to simplify and automate Grid service composition activities by delegating the decision-making and service orchestration tasks to external autonomous, intelligent systems, such as agents. Two key challenges arise in this attempt to develop a practical system that addresses this vision. The first challenge is elaborating the key properties of service workflows and the decisions that these agents are required to assist. This information is needed for design and implementation of the internals of the agent mind so that they can perform the decision-making tasks. The second challenge is designing and implementing the run-time interfaces between the workflow and agent environments to enable the exchange of information between them. This chapter also investigates how the framework for abstract workflow composition is extendable to support semantic annotations as well as identifying the components and interactions for an agent-based Grid-workflow architecture required for semantic workflow resolution. The work
conducted in this chapter is evaluated using the technologies and scenarios that are part of the ARGUGRID EU project.

6.1 Requirements for Automation

A number of challenges arise when defining and mapping semantically annotated workflows to concrete implementations. Many of these challenges relate to the properties associated with service composition constraints and models. In order to address these challenges a framework encompassing workflow systems, Grid middleware and intelligent systems must address the following requirements:

- **Semantic Descriptions**: Workflows need defining at a high-level using abstracted components. As identified in the previous chapter, semantic annotations help to normalize the search queries to prevent mismatches during service searching and can add value to abstracted, or semantic, nodes. These semantic nodes with descriptions can be composed together to build semantic workflows.

- **Performance Estimation**: Intelligent systems require various models in order to optimize the service execution, e.g. minimize cost or execution time. These models can range from individual service models with respect to QoS and fidelity, to workflow patterns for
determining the ordering of, and dependencies associated, with service executions.

- **Decision-Making:** Intelligent systems are required to select the best services to address the semantic workflow. These requirements must be specified by the workflow author and provide enough information for agents to perform their resolution functions. Agents can perform selection using the fidelity or QoS attributes of a service, as well as the service’s role in the overall workflow.

- **Interfaces to enable automation:** Introducing intelligent systems to the SOA framework requires extra interfaces to support communication between the different components, e.g. workflow systems, Grid middleware, agents, registries. Enabling automation requires brokering systems and translation tools that handle the exchange and resolving of data format issues between the components.

Further, more advanced challenges exist. Intelligent systems must ensure that an automatically generated implementation avoids illegal conditions such as deadlocks and live locks, for example. Similarly, the security policies governing specific data may impose further restrictions on the validity of the workflows. These issues are beyond the context of this thesis.
6.2 An Extended Architecture to Support Discovery Agencies

A Grid-workflow architecture requires extra components to support intelligent systems. These extra components must address the four requirements outlined previously. The semantic workflow support described in this chapter extends the Abstract Handler definition in the previous chapter.

6.2.1 Conceptual View

The following components are required in a Grid-workflow architecture to support intelligent-system based decision-making:

- **Workflow Systems**: for specifying semantic workflows and for co-coordinating the execution of workflows composed of remote services. Workflow systems have to be extended to support semantic descriptions and for enabling submission of semantic workflows to the agents for resolution and the retrieval of concrete workflows.

- **Semantic Annotations/Ontologies**: for describing and annotating both nodes and workflows. Ontologies provide the terms required for service matching and QoS negotiations.
• **Intelligent Systems**: for performing the service selection and composition tasks automatically on behalf of the user. The designs of the internals of intelligent systems need to support these tasks and support potential negotiations with the service provider around SLA terms.

• **Grid Middleware**: for hosting discoverable services that are executable from a workflow system. Each service published in a registry may have an associated SLA describing the non-functional properties associated with its execution for a particular user.

• **Semantic Services Registry**: to hold semantic descriptions and properties of available services that allow for service discovery, reasoning and selection.

• **Translation Tools/Brokering System**: for bridging the information exchange gap between the components of the framework as each typically uses its own internal representation.

• **SLAs**: Service Level Agreements state properties of a service, which describe the terms and conditions that surround the service execution.
Figure 6.1 shows a conceptual overview of how the components fit together from a workflow author's perspective, with agents playing the part of intelligent systems. The author constructs a semantic workflow and submits it to a requestor agent, which represents and performs decision-making on behalf of the workflow author. The workflow system submits the workflow to the requestor agent via a requestor broker, which houses the translation tools that converts the semantic workflow to the agent language. The requestor agent then communicates with other service provider agents, which are associated with hosted Grid services. Each service provider agent that is contacted searches for matching services in its associated registry via
Moving Towards Automated Service Composition Through the Use of Intelligent Agents

a provider broker. Discovering a matching service initiates negotiations between the requestor and provider agents over the execution properties. After successful negotiation, the requestor agent receives the service endpoint, the negotiated properties associated with the execution and any other additional information. The requestor agent repeats this process until it finds all the required services and begins constructing a workflow. This process may require several iterations of querying the registry and services until the agent has successfully composed a viable workflow description. The requestor agent sends the completed workflow description to the requestor broker, and retrieves and populates SLA templates with the negotiated properties. The requestor broker then submits the workflow to the workflow system, which renders the nodes and edges in the workflow client for execution. If the agent fails to construct a viable workflow description, it may seek further information from the user.

6.2.2 Semantic Annotations and Ontologies

Each abstract node must be annotated using a description language for defining the goals and sub-goals. A single attribute can be described in many ways, or a description can be used to describe two or more different attributes. Ontologies normalize the descriptions, providing two advantages: the first eliminates ambiguity of what is being described; the second improves service searching and term matching with possible values in registries, leading to better results.
Service requestors are able to query a service registry to discover and select services that fit their needs. Within a Discovery Agencies-supported SOA architecture, a registry responds by providing the requestor with details of how to connect to, and invoke the required services. In general, such registries can be regarded as the knowledge base used to store information about remote Grid services. Different registry implementations may hold different types of properties-based information regarding the available services, including:

- **Service functionality**: What the service does or its capability, i.e. a description of the high level functionality of the service itself.

- **Service input/output parameters**: A description of the input and outputs including their data types, names, metadata, etc.

- **Service invocation mechanisms**: Details of the access and invocation interface, the service bindings, endpoints, etc.

- **Service properties**: Details of the functional and non-functional properties of the service.
Note that all the service information does not have to be presented in a single registry and can be split amongst different registries depending on the implementation.

### 6.2.4 Brokering Systems and Translation Tools

From a requestor's perspective, brokers are required between the workflow system and the agent world to translate workflows between the two systems. From a provider's perspective, brokers are required between Grid middleware registries and the agent world to translate agent workflows to search terms for service look up. Either different brokers can be implemented for the requestor and provider perspectives or multiple instances of a single broker can be used which provides both the requestor and provider functionality. The latter option is preferred, as it allows requestors to be providers and limits the amount of duplication and development required to implement different brokers. The broker must perform a range of functions from language translations to workflow planning.

**Requestor Translations**

The broker has the task of taking the abstract workflow with annotations and translating it to a language understood by the agent world. The workflow language (usually in a XML format) consists of a number of details relevant to the workflow system. These details, such as workflow history and appearance in the workflow client, are not required by the agents and do not
Moving Towards Automated Service Composition Through the Use of Intelligent Agents

contribute to the service selection and negotiation process. The broker must separate this “housekeeping” information in the XML from the nodes, with annotations and edges between them. The information extracted is in two separate languages, the workflow language and the description language. The broker must provide two translational tools to covert both of these to the agent language.

Provider Translations

On the provider side, the broker must take the agent language and extract the attributes and goal descriptions in order to search over the semantic registries. Each matching entry found is translated into the agent language to allow the agent to reason and negotiate over the values.

6.2.5 SLAs

SLAs (Service Level Agreements) are contracts negotiated between the service requestor and service provider that determine the levels of each service. Information installed in an SLA can include non-functional properties and functional properties associated with the service. In some cases they can be tailored by the service provider for each user independently.
6.3 Semantic Levels of Abstraction

Identifying the components in a Discovery Agencies-based Grid-workflow framework determines what information is required at each stage in the semantic workflow resolution process. The abstract workflow implementation described in the previous chapter is not enough to support the more detailed workflow resolution processes required when using intelligent systems. Expanding the semantic annotations can address this problem by describing a node’s desired functionality, i.e. what computation the node should perform and the required inputs and outputs. Semantic annotations describe QoS properties not associated with the function of the node, but which govern its execution, e.g. expected execution time and price limitations that are usually stored in SLAs. A semantic workflow is composed of a number of semantically-described nodes, enabling the construction and refinement of the overall workflow logic from a top-down perspective. Typically, semantic workflow descriptions fit into the following levels of abstraction:

- **Goal**: At the highest level of abstraction, i.e. at the semantic workflow level, the user defines what the overall workflow is set out to achieve.

- **Sub-goal**: Moving top-down, the next level is at the semantic node level. This level defines both the sub tasks or intermediate steps in order to achieve the overall goal, as well as the order in which each of the tasks must occur.
- **Attributes:** The lowest level of abstraction describes the inputs, outputs and properties (or parameters) for each of the semantic nodes. These properties are either functional or non-functional attributes, specific to the function of the service or general constraints not dependent on the function respectively.

As mentioned in the previous chapter, before the execution of a semantic workflow, each semantically described node needs replacing, i.e. binding to an existing service. Using only the higher levels of abstraction will typically provide a wider search space when selecting services to implement the workflow. Using descriptions that are more detailed will typically restrict the number of matching services. Note that adding priorities or rankings to service descriptions guides the search for specific results, e.g. finding the solution that is the cheapest, or the solution that will provide the quickest results. Also, note that selected services must respect other constraints defined by the workflow structure itself. For example, for each workflow node, the input and output types of the real service implementing it must match the node description to ensure the production of a viable executable workflow graph.
Goal
Sub-Goal Sub-Goal Sub-Goal Sub-Goal
Functional Attributes
Non-Functional Attributes
Sub-Goal

Figure 6.2: Levels of abstraction required for semantically describing workflows

Figure 6.2 shows how the different levels of abstraction fit within the context of a semantic workflow. A single goal definition consists of four ordered sub-goals. Each sub-goal is broken down into functional attributes (associated with the ports and parameters) specific to the operation required, and non-functional attributes which govern the execution. Functional and non-functional attributes are paramount to the decision-making process with respect to selecting services to address a desired goal or sub-goal.

Functional Attributes
Functional attributes describe the requirements for each sub-goal and score how close a service matches to the required functionality. Functional attributes are not as generic as non-functional attributes, which are applicable to most if not all sub-goals. In order to resolve functional attributes, a form of normalization is required, as the names and values of
attributes can come in various representations, e.g. a string or set of longitudinal and latitudinal coordinates could represent a location attribute, thus ontologies, must keep these values consistent.

**Non-functional Attributes**

Non-functional attributes are associated with the execution and are independent from the actual function. The key non-functional attributes required for service selection are execution time and cost of a service.

### 6.4 Service Composition Models

The typical methodology for building any form of workflow, but especially data flows, is to start from the first task and progress towards the final task. Service co-ordination is resolved in this manner. By analyzing how the services chain together, and looking at the entire workflow rather than individual services, however, it is possible to optimize each workflow during resolution. For example, executing two independent services in parallel may result in an overall quicker execution time compared to executing them sequentially. The composition of a workflow can have a significant effect on the costs associated with the workflow and fidelity of the concrete workflow returned. Using an expensive service of higher fidelity early on in a workflow, for example, may reduce the tasks required later on and thus leads to a cheaper workflow overall compared to using a lower fidelity service. In such
situations, it is necessary to look beyond the matchmaking of individual services in a workflow and analyze the composition of services.

As mentioned in the previous chapter, when composing services, one service may not directly connect to the next service. This mismatch can occur due to a number of reasons, such as conflicting input and output types or missing information. Shims are intermediate services that provide the missing link between two services, as shown in Figure 6.3. Although conceptually simple, shimming services present a number of issues. Shimming services may add additional computing costs to the overall workflow, may increase the chance of failure of the whole workflow, or may affect overall workflow reliability or fidelity of results.

![Concrete Workflow Diagram](image)

**Figure 6.3: Using shimming to bridge the gap between services**

The following examples demonstrate methods used for resolving mismatches between services and resolving goals, which do not map to a single service instance. When situations like these arise, multiple solutions
resolve the mismatch, each with an associated cost (monetary, computationally, time or fidelity of results) to minimize as much as possible.

6.4.1 Resolving Two Mismatching Services Located on the Same Grid Server

Figure 6.4 shows two sub-goals resolving to two services, one for each goal respectively. Both services reside on the same server and are the best options for resolving the semantic requirements. The services cannot be connected together to produce a complete, concrete workflow, however, because of the type mismatch between the output of the Service A and the input of the Service B, and because Service B is missing certain information.

![Diagram of two abstract descriptions resolving to two mismatching services](image)

**Figure 6.4: Two abstract descriptions resolving to two mismatching services**

In Figure 6.4 and the following figures, Service A and Service B are different services with the subscript number denoting different implementations of the
same service in terms of types, attributes, inputs required and outputs returned.

**Service Replacement**

Figure 6.5 shows two ways of resolving the service mismatch shown in Figure 6.4. The example on the left replaces Service A with a different implementation that satisfies the input requirements of Service B, i.e. they have the same object type, and for more specific matching, the same name. This replacement may bring disadvantages, e.g. the new implementation may have a higher cost or lower fidelity. The decision on whether to replace service A or B to resolve the mismatch depends on which change would provide the least compromise or negative impact on the overall workflow.

![Diagram of service replacement](image)

**Figure 6.5: Changing one service or both services in order to resolve mismatches**

The example on the right in Figure 6.5 demonstrates the replacement of both service A and B to resolve the mismatch. Different implementations of both services are required if there is no possibility of resolving the mismatch by
changing just one of the service implementations, or if changing one of the services heightens the negative impact when compared to changing both services.

**Adding Shims**

A shimming service bridges the gap between Service A by providing extra functionality required by Service B that Service A does not provide or by converting the output data from Service A to the format required by Service B. Unlike the replacement methods described previously, using a shimming tool preserves the most applicable services by not compromising on service choice. Adding shimming services may incur additional monetary and computational costs, and possibly increase the overall workflow execution time. In some cases, more than one shim may be required to resolve the mismatch between two services as shown in Figure 6.6.

![Figure 6.6: Adding shims in order to resolve a mismatch between two services](image-url)
6.4.2 Resolving Two Mismatching Services Located on Different Grid Servers

The previous examples assume that all the required services exist on the same Grid server; contrarily, the most suitable services may exist on different servers. In such situations, analysis of further attributes, such as data transfer, produces optimal workflows. Figure 6.7 shows two sub-goals resolving to two services existing on different Grid servers, and like the previous scenario, a connection between the two services cannot be established.

![Diagram showing two sub-goals resolving to two services on different Grid servers]

**Figure 6.7: Example of two mismatch services located on different Grid servers**

**Service Location**

The service replacement and shimming models described previously are applicable to the scenario in Figure 6.7, but when service location now becomes an important factor in the resolution process, consider the example shown in Figure 6.8.
Figure 6.8: The affect of service location on the decision making process

Figure 6.8 shows two options for resolving the service mismatch show in Figure 6.7. Both replacements for Service B have the same fidelity rating. Service B in the left workflow has a quicker execution time and resides on a different Grid server than Service A. Service B in the right workflow has a slower execution time and resides on the same Grid server as Service A.

Which service B implementation to use depends on which workflow has the better performance: the service execution and combined data transfer in the left workflow, or the service execution in the right workflow. Service locality decisions also apply to the shimming process as the location of the shim can affect the overall workflow execution time and performance.

**Data Size**

In situations where data transfer is unavoidable, the aim with respect to execution time is to reduce the overall file transfer time that takes place in a workflow. Figure 6.9 shows an example with Service A and Service B
residing on different services and requiring a shimming service. The required shim is available on the same server as Service A and the same server as Service B. The shim adds additional information to the output of Service A and hence the output of the shim is larger compared to the input. Consequently, using the shim that resides on Server B will improve the performance compared to the shim on Server A, due to the transfer of a smaller file.

![Diagram](https://via.placeholder.com/150)

**Figure 6.9: Factoring the location of a shim service to resolve service mismatching**

**Type matching**

Curcin in [25] describes how type matching affects the composition process, particularly for complex, structured data types such as tables, taking into account column metadata as well as general table metadata. For example, if the second service takes an integer, the first service must return an integer to satisfy the type constraints. Note that with the Grid examples presented in this chapter, the Grid services are all file based, therefore type constraints
are not taken directly from the data type. Instead, the type matching process uses type information specified in the service metadata information.

6.5 Models for Service Selection

The previous chapter introduces the concept of automated service discovery using a semantic broker (such as the one provided by NEC) to choose between services. The service selection process focuses only on FE-EI services using QoS constraints only. Matching other forms of service (FE-DI, FS-EI, FS-DI) within the context of an agent framework, requires a range of models for service selection based on the functional and non-functional attributes.

6.5.1 Models for Attributes

The following models describe the functional and non-functional attributes for a service.

**Functional Attributes**

Cardoso's model for fidelity [17] addresses functional attributes, scoring how well a service matches the requirements and how important each attribute is. Each service $s$, has a fidelity measure $F$. $F$ is a vector composed of functional attributes $(F(s).a)$, where $a$ is an attribute. Not all attributes carry the same importance, Cardoso thus adds a weight attribute $w$, to each
attribute. For \( n \) number of attributes, the sum of the weights equals one.

\[
F(s) = |F(s) \cdot a_i \cdot w_i| + |F(s) \cdot a_j \cdot w_j| + |F(s) \cdot a_k \cdot w_k| + \ldots + |F(s) \cdot a_n \cdot w_n|
\]

**Non-Functional Attributes**

The following models address the non-functional attributes, namely execution time and cost.

- **Execution Time:** Cardoso [17] presents an execution time model that indicates how long the process is required to take. This equation takes the form of a range of values. In general, the service time \( T \), is the actual execution time \( ET \), in addition to any probable delay time \( DT \), or queue time \( QT \).

\[
ET(s) \leq T(s) \leq ET(s) + DT(s) + QT(s)
\]

Note that these terms are associated with the service only and do not take into account any data transfer time or additional time associated with the workflow, which this chapter covers later.

- **Cost:** Cost represents either monetary or computational costs for service execution; monetary in terms of the charges for invoking a service, and computational in terms of the number of CPUs used or memory allocated for performing the execution. In Cardoso’s model [17], the service cost \( C \), for a service equals the sum of the realisation cost \( RC \), associated with the runtime execution of the task and enactment costs \( EC \) associated with the management of the workflow.
system and instances of monitoring [18].

\[ C(s) = RC(s) + EC(s) \]

Note that non-functional properties are negotiable between providers of services and service requestors. For example, a user may be willing to pay a higher price to reduce the service execution time. Similarly, the user may also be willing to trade-off one of the functional properties (e.g. quality of results returned by a service) with other functional or non-functional properties.

### 6.5.2 Workflow Arrangement Models

The previous models only considered individual service selection. However, in order for intelligent systems to resolve workflow service composition must also be taken into consideration. Experiments involved executing GRIA v5 services from an InforSense v4 installation using the DTE-M based GRIA plug-in to determine the variables for each model. These results are based on image processing services with the execution results broken down into their various processes (upload, execute and download). This information was captured by InforSense task manager and identified in the logs. Table 6.1 shows the average results for executing a single service with varying file sizes.
<table>
<thead>
<tr>
<th>Input File Size (Mb)</th>
<th>Workflow Initialization (s)</th>
<th>Data Upload (s)</th>
<th>Job Execution (s)</th>
<th>Data Download (s)</th>
<th>Total Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>14.5</td>
<td>41.6</td>
<td>2.8</td>
<td>59.4</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
<td>18.2</td>
<td>40.5</td>
<td>3.6</td>
<td>62.7</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>30.7</td>
<td>47.7</td>
<td>6.4</td>
<td>85.3</td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>41.7</td>
<td>55.7</td>
<td>7.0</td>
<td>104.9</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>72.4</td>
<td>58.3</td>
<td>17.1</td>
<td>148.3</td>
</tr>
</tbody>
</table>

Table 6.1: Breakdown of workflow execution times for a single service accessed using the DTE-M mapping

Similar experiments and analyses were performed for sequential, parallel and looping models in order to determine the models that follow.

**Sequential Execution**

Sequential Execution is one of the most basic forms of workflow modelling. One service leads to the next in a simple chain, with each service dependent only on the ones preceding it. The following models are based on a DTE style node mapping:

- **Time:** Considering a simple workflow that executes one service that requires an input (as shown in Table 6.1), the execution $ET$ for a service is broken down into workflow initialization $WI$, data upload $DU$, job execution $JE$, data download $DD$ and any additional delay time $DT$. 
The next model considers executing $n$ number of services together in a sequential chain. Note that this model assumes each service is running on different Grid server, hence data upload and download operations are required between each job execution.

$$ET(s) = WI(s) + DU(s) + JE(s) + DD(s) + DT(s)$$

If the services exist on the same server and the download and upload operations between services is removed, the following model can be used:

$$ET(s) = WI(s) + DT(s) + \sum_{i=1}^{n} (DU_i(s) + JE_i(s) + DD_i(s))$$

Note that if a workflow consists of services that run on the same and different servers then both models can be combined.

- **Cost:** Cost $C$ in a sequential execution model is the summation of the costs for executing each individual service.

$$C(s) = RC(s) + \sum_{i=1}^{n} EC_i(s)$$

- **Fidelity:** Fidelity becomes even more unpredictable and difficult to model when considering workflow execution patterns. In the examples concerning only two services, fidelity is measured individually for each service independent of the services before and after it. In some cases this is true, however in certain situations the fidelity of a service is
affected by the quality of the data coming in and may have a knock on effect, reducing the effectiveness of each service that follows it.

Parallel Execution
The advantage of parallel execution is that it may reduce the overall execution time of the workflow by running at the same time. However, by running services in parallel, there is a chance that two or more services may end up sharing the same resource (e.g. slower uploads due to two or more files being sent at the same time) and this could affect whether the non-functional attributes are met.

- **Time:** When modeling the execution time for services in a parallel, each branch $PB$ is considered separately. This assumes the branches are independent, i.e. each branch has no bearing on the other, such as not affecting the resources available to another branch or accessing different services.

$$PB(s) = DT(s) + \sum_{i=1}^{n} (DU_i(s) + JE_i(s) + DD_i(s))$$

The aforementioned model for calculating $PB$ is based on a branch running services sequentially on different servers. The total time for a parallel workflow $ET$ is the workflow initialization $WI$, delay time $DT$, any delays due to parallelization $PD$, and the time it takes to execute the longest parallel branch $PB$.

$$ET(s) = WI(s) + DT(s) + PD(s) + \max_{n}([PB_1(s), PB_2(s), ..., PB_n(s)])$$
• **Cost:** The cost for executing branches in parallel is the sum of all the costs for executing each individual service as all services are still charged for.

**Looped Execution**

The following models consider two forms of looping. The first model submits one set of inputs and runs multiple job instances that use this set. The second model uploads a new set of data inputs for each iteration of the loop.

• **Time:** Like the parallel models, each loop sub workflow $L$ can be considered separately. The first of the following models considers uploading a new data set each time, while the second iterates over one set of data.

\[
L(s) = DT(s) + \sum_{i=1}^{n} (DU_i(s) + JE_i(s) + DD_i(s))
\]

\[
L(s) = DT(s) + DU_1(s) + \sum_{i=1}^{n} (JE_i(s) + DD_i(s))
\]

The total execution time $ET$ for a workflow consisting of loops is the workflow initialization $WI$, loop initialization $LI$, loop $L$ and any delay time $DT$.

\[
ET(s) = WI(s) + LI(s) + L(s) + DT(s)
\]

• **Cost:** The cost for executing loops is the same as each loop is charged for individually.
6.5.3 Using Models

Given the nature of SOA Grid services, it is difficult to predict the actual values for workflow execution since there are many factors that affect performance. While SLAs outline the conditions governing the services invoked, there are no measures to ensure that the workflow execution meets particular standards. For example, data transfer is heavily dependent on the bandwidth available and the amount of traffic. Workflow overheads are dependent on the resources available to a workflow system, which can vary depending on the machines used, resources available to the workflow system and the number of workflows running simultaneously.

These models are therefore not used to predict the actual time and costs of workflows, but rather to guide the decisions made. For example, based on parallel and sequential models, executing two long-running services in parallel would reduce the overall workflow time compared to running them in sequence. Service and composition models can either be hard-coded into intelligent systems or fed to them dynamically. The former method has simpler implementation while the latter uses more up-to-date models for performing decision-making.
6.5.4 Workflow Arrangement Models Summary

Three different workflow models have been presented which demonstrate how the arrangement of services can be affected. Fidelity is used to measure the functional equivalence and can range from simple attribute matching to complex matching through the use of weights. In terms of non-functional properties, cost is not affected by the model used, for example executing a set of services in parallel or sequentially has the same monetary costs. Execution time is greatly affected by the model chosen, outlining an importance in defining these models. Workflows can generally be summarized using one or a combination of these arrangement patterns, as shown in Figure 6.10.

![Figure 6.10: Identifying the arrangements within a complex workflow](image)

In Figure 6.10 the first parallel segment contains three branches: the first branch invokes only a single service, the second invokes a sequence of
services and the third consists of a loop. The second parallel segment contains a single service execution and another parallel execution. By breaking down workflows into separate workflow constructs, it is possible to apply the workflow models to obtain the overall workflow costs.

6.6 Semantic Workflow Composition Support

Supporting the conceptual view described previously requires an identification of particular workflow system and agent (or other intelligent system) details. The choice of workflow system and agent environment greatly impacts the implementation of the components to support the framework.

6.6.1 Semantic-Based Grid-Workflow Architecture

Figure 6.11 shows the layered architecture to support automated resolution of semantic workflows.
Like the Abstract handler, the semantic handler works in tandem with the Grid handler. A brokering system handles communication between the workflow and autonomous agent environment using translational tools. Registries house information about published services for discovery and negotiation. This chapter discusses the design implications for the Semantic handler and the supporting components of the architecture to support communication with intelligent agents.
6.6.2 Semantic Node Definition

Like the abstract node, the semantic node requires at least one output and a configurable interface to allow for the addition of more input and output ports. With the semantic node, a finer level of granularity is required when specifying the requirements due to a more complex and accurate matchmaking process. Therefore, rather than just manipulating the number of ports on the semantic node, the user must be able to annotate each port in terms of the required type and an ontological term describing the data sent to or from the port. Beyond ports, the node must support the definition of node parameters, which map to the functional attributes of the required sub-goal, using ontological terms and type descriptions. The semantic node must also provide support for additional ontological terms, which define the operation of the node, and non-functional attributes, which determine the execution conditions.

6.6.3 Semantic Support Interfaces

The semantic node and workflow system require the following support interfaces for submitting (via the brokering system) semantic workflows to the agent world, and returning concrete workflows from the broker respectively. These interfaces are adapted from the support interfaces for the abstract handler.
Node Annotation Support

This interface supports the handling of semantic annotations for each semantic node. The handler must save and persist a semantic description associated with each instance of the semantic node. Extensions to this interface can support multiple annotation languages if required, as well as provide a means to validate and check the structural integrity of the annotations.

Semantic Workflow Submission

This interface handles the submission of the underlying, XML-based description of the semantic workflow, with annotations to the brokering system for resolution.

Concrete Workflow Retrieval

The concrete workflow retrieval interface handles the incoming concrete workflow provided by the brokering system. Authoring time checks performed on the returned workflow highlight any potential issues, such as missing compulsory parameter values, or unresolved node connections.

SLA Support

Each service returned in concrete workflow may have an associated SLA outlining the execution conditions negotiated by the agents for that service. This interface is responsible for rendering the SLA in the client for the user to review, and for submitting the SLA to the appropriate Grid server at authoring.
or execution time, indicating that the user agrees with the specified terms. Depending on the workflow implementation, the Grid handler, rather than the semantic handler, provides these SLA support functions.

### 6.6.4 Semantic Interactions

The interactions between the components identify the flow of information between the architectural layers.

**Submission of Semantic Workflow**

The interaction diagram in Figure 6.12 shows the process for submitting a semantic workflow to the broker for resolution by the agents. The semantic workflow consists of semantic components that utilize the Node Annotation Support object. The annotated workflow uses the submission object to pre-process the XML workflow representation, submit the workflow to the broker, and wait for a response. The broker translates the workflow into a language understood by the agents and submitted to the requestor agent for service discovery and negotiation.
Moving Towards Automated Service Composition Through the Use of Intelligent Agents

Retrieval and Rendering of Concrete Workflow

The broker sends the concrete workflow, produced by an agent, to the workflow planner for transformation into an executable workflow. The semantic handler then receives the executable workflow, where pre-processing occurs before the workflow is rendered in the client. The workflow author or domain expert can then run the executable workflow. Figure 6.13 shows the interactions for this process.

Figure 6.12: Interactions for submitting an abstract workflow to the broker
6.6.5 Implementation of Semantic Components

With semantic workflows, the various components that interact with the agents must also be implemented, i.e. registries, brokering system and translation tools. The specifics of these components depend on the agent environment used. The implementation described here centers on the components and environment used with in the ARGUGRID project [57].

Agent Environment

The ARGUGRID multi-agent environment consists of two key components, GOLEM and MARGO. GOLEM [15] is a cognitive agent environment middleware which is an evolution of the PROSOCS [37] system. MARGO [78], which provides the agent mind written in Prolog, implements a reasoning engine based on the argumentation framework of the CaSAPI [49]
system, a general purpose tool for assumption-based argumentation. It should be noted that within ARGUGRID, GOLEM agents can discover, communicate and negotiate with other agents supported by an underlying peer-to-peer platform, PLATON ++ [73].

**Semantic Annotations and Ontologies**

The goals for each semantic node and workflow are described using the WSMO language with specialized ontological terms for each domain supported. WSMO was chosen since it addresses both the service requestor and provider perspectives. Regarding the service requestor view, WSMO provides a conceptual model for semantic web services (and applicable for Grid). These include:

- **Requested Service Capabilities** for addressing functional requirements;

- **Requested Interfaces** for describing the communication behavior for consuming a service;

- **Restrictions/Preferences** which can be applied to both functional and non-functional properties.

Regarding the server provider view, WSMO provides Interfaces for describing the usage, and Capabilities for describing the function, how to interact with the service, and how services interact with each other. For
ARGUGRID, Earth Observation scenarios spawned ontologies with ARGUGRID-specific concepts.

**Service Registries**

Registries provide the knowledge base for agents, allowing them to search and locate services matching their requirements, and service parameters for negotiation. Deploying a single or multiple registry implementation depends on the chosen method, service information to be stored, and agents used. This example incorporates three registries: an agent registry, a semantic service registry and an SLA registry:

- **Semantic Registry:** The semantic registry stores information about each service published by the service provider. It consists of endpoints for each service, a description and the publishing organization. For each service there are lists of functional and non-functional attributes that map to the values specified in the ontologies. For each attribute, the registry provides a name, value or range of values, and type is provided.

- **SLA Registry:** The SLA registry contains GRIA templates whose values can be replaced with the values negotiated by the agents. The values in the SLA template match the non-functional attributes specified for all services, namely, time and cost. To match the
template to the service, the service endpoint is also included in this registry and used as the primary key.

- **Agent Registry**: The agent registry contains information associated with the agents and allows the agents to discover one another. The registry also gives the organization with which the agent is associated, which links to the publishing organization in the semantic service registry.

Three registries were chosen rather over a single global registry because each one contains contextually different information associated with different components in the framework.

**Translation Tools and Broker**

The implementation of the ArguBroker component is based on a modular design and its own internal representation of workflows and service descriptions. These features enable it to be flexible and independent from the workflow system, agent environment and semantic registry, thus allowing specific implementations to be interchangeable. The component also implements the interfaces for exchanging information with the workflow system, the services, the service registry and the agent, as well as a number of translators. The internal structure of ArguBroker is shown in the Figure 6.14. ArguBroker itself is accessible through Web Service interfaces to simplify its interaction with a variety of systems.
Translations from the workflow to the agent involve extraction of information from XML documents such as DMML, WSMO annotations and SLA templates based on templates before populating PROLOG templates with the required values. The opposite process occurs when moving from the results return by the agent to the workflow language.

ArguBroker supports the following translators:

- **DMML2Prolog and Prolog2DMML Translators:** DMML2Prolog translates the semantically annotated workflow representation (in this case InforSense’s DMML) into the representation used by the agent...
Moving Towards Automated Service Composition Through the Use of Intelligent Agents

(Prolog). Prolog2DMML translates the concrete workflow returned by the agent into the workflow system representation.

- **WSMO2Prolog and WSMO2Prolog Translators**: WSMO2Prolog translates user annotations in the semantic workflow expressed in WSMO into Prolog for the agents to reason over. It also translates WSMO service descriptions from the registry into Prolog. Prolog2WSMO translates the Prolog representation of service properties to WSMO primarily for querying the semantic registry.

- **SLA2Prolog and Prolog2SLA Translators**: Converts between the SLA representation used by the service provider (the GRIA SLAs in this case) and the representation used by the agents (Prolog).

- **Executable Workflow Generator**: used for adding InforSense workflow engine-specific information, and responsible for first taking a partial DMML representation of a workflow representation to create a complete DMML workflow executable in the InforSense workflow system.

Note that the structure of ArguBroker and its modular implementation allows for the addition of further workflow and semantic description translators, or changes in accordance with the implementations chosen.
6.6.6 Semantic Handler Implementation

The following semantic handler implementation is based on the components described earlier and thus are built with WSMO descriptions and the ArguBroker translation tools in mind.

Semantic Node

The semantic node extends the implementation of the abstract node. Using available interfaces in the InforSense API, a configurable interface allows the user to control the number of ports and parameters associated with the node. For each port, a type or range of applicable types is specifiable along with an ontological description. Each port also may be either compulsory or optional. Likewise, each parameter can have an ontological description and type set. Like the abstract node, the semantic node has no process method associated with it and cannot be invoked as a traditional data flow. Note that this particular implementation supports the addition of free WSMO annotations to describe the non-functional attributes and the overall purpose of the node, i.e. the sub-goal. The translators are responsible for parsing the port and parameter information for the semantic node and converting them to WSMO, adding to the descriptions.

Annotation Panels

The annotation panels (shown in Figure 6.15) are interfaces in the workflow client that allow the end user to add WSMO annotations to each node. In this
particular case, the workflow author uses ontologies and a WSMO editor to create the WSMO descriptions, before copying and pasting them into the panel. Note that syntax checking or even a custom WSMO editor can be built into the panel to aid the end user if necessary. By keeping the descriptions as free text, any description language can be supported and only the underlying translation tools in ArguBroker need to changed. To improve readability, another panel can be created as part of the handler that takes the WSMO annotation, parses it and extracts the element and attribute names with their associated values, and displays them as a cleaned up list.

Figure 6.15: Semantic node with associated annotations. The annotations are entered as WSMO documents and parsed for clearer viewing
Abstract Workflow Submission

After the Abstract workflow creation and annotation, the workflow needs to be sent to ArguBroker. ArguBroker offers a web service interface; therefore, the handler contains stubs and interfaces generated from the endpoint. These objects are packaged within the handler and accessed by the workflow submission object. This object invokes the workflow resolution method. The endpoint to the ArguBroker web service is added to a configuration file, which can be modified and validated through the portal interface. This allows the endpoint to change without regenerating and recompiling the stubs. Changing the method or methods, would require stub regeneration and recompilation. A menu link invokes the workflow resolution method for workflow execution. This invocation reads the DMML into a string object and submits it to the web service.

Executable Workflow Retrieval

The graph produced by ArguBroker is returned as a String from the workflow resolution method call. The SLAs stored in this workflow contain escaped opening and closing tags when returned, so a find and replace method restores the tags for correct visualisation in the client and for DOM object construction for submission to the GRIA server. To aid usability, the semantic handler contains methods, which throw exceptions from the workflow client, before the workflow is executed, highlighting which nodes require their SLA to be submitted.
SLA Support

The InforSense node editor SLA panels display the returned SLAs. (Figure 6.16). Like the annotation panels for the WSMO annotations, one shows the actual SLA XML while the other parses the XML to display the negotiated attributes in a clear and readable manner. A toggle added to the SLA panel allows the end user to accept or reject the SLA. The node throws an exception and cannot execute if the user rejects the SLA. Accepting an SLA submits the SLA object to the GRIA service call based on a method provided by the GRIA API. It is possible to build SLA submission functionality into the GRIA handler, thus removing the GRIA library dependency, but as this example does not need general GRIA service execution, the semantic handler implements the SLA submission functionality instead. The endpoint for the submitted SLA is stored in an instance of an SLA object in the semantic handler that is called when the corresponding GRIA service is invoked.
Figure 6.16: Returned concrete workflow with an associated SLA. The workflow is not executable until the SLA is accepted and submitted to the Grid server.

6.7 ARGUGRID Case Studies: Earth Observation

Due to the complexity of the automated decision making processes, multiple scenarios evaluate the different aspects of the framework updates to support automation.
6.7.1 Environment to Support Case Studies

On the service requestor side the InforSense v4 workflow system, with the GRIA and semantic handlers, builds abstract workflows and execute concrete GRIA workflows respectively. The DTE-M mapping models the GRIA handler in this case. An implementation of ArguBroker performs the translations between the workflow system and agents. A service requestor agent communicates with other agents in order to negotiate services and build the workflow. On the service provider side reside two GRIA v5 servers. Each server has a different set of non-functional attributes associated with its services. In total, fifteen services are deployed across the servers (five for each service: getImage, typeConversion and pollutionMap). Each implementation of a service has a different set of functional attributes or inputs/outputs associated with it. For each GRIA server, a service provider agent is responsible for service negotiation. The agent interacts with two registries: the service and SLA registries, which publish service details (i.e. functional and non-functional attributes) and SLA templates, respectively. GOLEM, MARGO and CaSAPI implement all the agents, which communicate using the PLATON++ P2P system.

An ontology specific to the Earth Observation scenario specifies the terms usable in WSMO documents, which can be added to the nodes. A workflow author uses an external WSMO editor to browse the ontology and construct the annotations. These documents are then copy to the nodes. Alternatively,
the workflow author can write the WSMO annotations directly to the node, without checks present to ensure the document is valid. The workflow client may also support editing WSMO annotations and performing error checks directly, via custom node interfaces.

A range of deployed services cover all possible scenarios from the three main service selection criteria in Table 5.1: interfaces, functionality and QoS. Expected and optimized results are known for each scenario, in order to ensure the agents are performing the correct decisions.

6.7.2 Workflow Resolution Process

The following steps cover the end-to-end scenario of defining an abstract workflow and submitting an executable workflow. First, the workflow author uses the InforSense client to build a workflow consisting of semantic nodes. By default this node cannot be executed until annotations are added and the ports are configured with connections between them. For each node and the workflow a set of annotations are produced representing the fidelity, ports and QoS constraints. When all semantic nodes are annotated and the connections between them defined, the workflow is executable.

When the workflow executes, the underlying DMML graph submits in XML format to ArguBroker. This graph contains specific information about the nodes and any connections, along with the WSMO annotations. In order for
the message to transfer correctly, the XML tags must be escaped and then reinstated once the message arrives at ArguBroker. ArguBroker takes the DMML and removes the housekeeping information, namely node history, date of workflow creation, etc. The remaining workflow information is translated to PROLOG for use with the agents. ArguBroker then submits the PROLOG workflow to the Service Requestor Agent that represents the workflow user.

The Requestor agent analyses the workflow and sets to resolve the services. Using the PLATON++ P2P environment the Requestor Agent contacts Provider Agents. Each Provider Agent receives a service request, processes it and submits it to its own instance of ArguBroker. ArguBroker receives the PROLOG service request and extracts the terms for querying the registry. If any matching services are found, the Provider agent notifies the Requestor agent beginning negotiation and decision-making over services. Once agents agree upon services and complete negotiation of the non-functional attributes, the Requestor agent builds a concrete workflow that determines the ordering of the chosen services and submits it to ArguBroker.

ArguBroker translates the PROLOG representation of a concrete workflow to DMML and uses the workflow planner to produce nodes, thus defining complete workflows, i.e. to ensure that each service includes execute, upload and download nodes as needed. ArguBroker handles this process, as it is specific to the implementation of the workflow system. After producing a
concrete DMML workflow, ArguBroker contacts GRIA registries to retrieve SLA templates, one per service. Each SLA template returned is populated with the values negotiated by the agents. The housekeeping information is then restored and the workflow is retrieved by the workflow system. As with the submission process, the web service interface is used and the DMML, as part of the SOAP message body, must be escaped and restored once it arrives at the workflow system.

The retrieved workflow renders in the client with the SLA information parsed and shows in the node panel for easy reviewing. The returned workflow can be inspected and edited before execution. Edits span adding manual local shims, building around the Grid flow returned, or removing services if requirements change. In order for the workflow to be executable, the SLAs must be accepted. SLA acceptance submits the SLA to the corresponding GRIA server and a reference persists. If a workflow author saves a workflow with submitted SLAs, the acceptance decision also saves to prevent SLA resubmission. When all SLAs in the workflow are submitted, the workflow can execute from end-to-end. Each invoked service uses its associated SLA.

### 6.7.3 Case Study 1: Service Discovery

The aim of the first case study is to test simple service selection based on service fidelity and QoS constraints. Only a single node is defined so there is no composition involved in the process. The service required must produce a
satellite image of a particular region taken at a specific time. Using the ontology a set of characteristics can be defined for the node annotation. Location is specified as a string specifying the town, country or continent. Date indicates when the image was taken and specified as a set of values, giving a range of possible dates. Narrowing the search range may reduce the number of hits the agent returns. Size defines the dimensions of the image returned using height and value. Resolution determines the quality of the picture in pixels. The output file format may be specified as JPG, GIF, PNG or BMP. These values and ordering determine the fidelity of the service. Separate QoS constraints also determine the preferred execution time and cost.

With the defined semantic node, location is set as the most important parameter, followed by resolution, size and finally time (represented by a broad date range). The QoS parameters in this first instance are open as the main goal is to achieve a high fidelity of results, regardless of the time and cost. The requestor agent contacts provider agents for matching services and caches potential matches. The agent then reasons over the services to find the best possible service for the end user. Where two matching services are found that have the same interfaces and fidelity of results, the agent then reasons over the next criteria, in this case the QoS parameters. The requestor agent negotiates with the provider agent for the best possible values and based on the negotiation accepts one of the services.
Moving Towards Automated Service Composition Through the Use of Intelligent Agents

As this particular node has only one output, the workflow planner within ArguBroker uses an execute job node and a download node to represent the workflow in DMML (Figure 6.17). This logic has to be built into the workflow planner, i.e. the determination that a lack of inputs to a service implies no upload data node is required in the workflow.

Figure 6.17: Returned workflow for getImage. No inputs so only a getImage and Download Data node are returned by ArguBroker

Changing the priorities of the properties results in a different service being returned. As date currently has the most importance, a lower quality image closer to the actual date is returned. Workflow authors may have to refine semantic workflow definitions, changing priorities in order to get the workflows they require.

In cases where the agents cannot fully meet the functional and non-functional properties, weights must help determine which service has the overall highest rating. Weights can declare that an image a few days earlier than expected is better than an image with lower resolution, for example. Weighting the options and pinpointing a reasonable solution is a tricky process. Ideally, a range of workflows should be returned to let the workflow
author make the final call on the workflow chosen, or even take fragments from each of the workflows to build what best meets his or her needs.

6.7.4 Case Study 2: Service Composition

This case study extends the first one by introducing a simple case of composition. Using the earth observation domain, apply a pollution map to the satellite image returned in the first case study, highlighting the pollution hotspots in a particular region. Two semantic nodes result and a connection between them indicates the ordering of operations. The first node comes from the first case study, and the second node is a placeholder for a service that takes the first image and applies a pollution map over it, returning a new image. The properties of this service are location and date, which determine the set of data for pollution density calculations.

For the first node, labelled as getImage, location is a priority as the image must be of the same region as the applied pollution map. Resolution is also high as the aim is to produce a good quality image. The date of the picture has the least priority, as the geographical layout of the region has not changed much over recent years. For the second node, pollutionMap, the location takes priority and date ranks high since the user is looking at a specific time scale. Figure 6.18 shows a semantic workflow containing the two aforementioned nodes.
The broker submits the workflow to the agent, and the agent looks to retrieve the best services based on the criteria. If the best services match in terms of interfaces then they can be composed together and sent back to ArguBroker for translation. Figure 6.19 shows the returned workflow. Given that each service is on a different Grid server, download and upload nodes are needed between each GRIA service execution node. If two adjacent services are from the same workflow server, the execute job nodes are connected directly by the workflow planner for basic optimization.

This scenario resolves each node individually with respect to the whole workflow. Analyzing the whole workflow has distinct advantages. Consider
the QoS attributes for each service, if cost is a priority then each service can resolve to the cheapest. However, if the priority is to get the best possible outcome within a certain price, then the decision making process becomes more complex. For example, if the best service for getImage is cheaper than the defined non-functional attributes, the pollutionMap service may supersede the requirements set out in the node, but still meet the overall workflow requirements. Of course, these decisions are more complex and lead to longer workflow resolution processes.

6.7.5 Case Study 3: Shimming

In the Service Composition case study, the output from getImage presumably feeds directly into the input of pollutionMap. In this case study, interfaces affect the service selection and composition process, and possibly require shimming. The semantic workflow outlined in the Composition case study applies here but with a restriction to retrieve only JPG format. When the workflow is submitted, the number of service matches reduces to only those with JPG format. Given that the getImage services return PNG files, and the pollutionMap services only take and produce JPGs, a service mismatch results.

In case of mismatches, the Requestor Agent looks at the interfaces for the getImage and pollutionMap services and outlines a new semantic definition for a service that can take the first format and produce the second,
essentially a data conversion tool. Any available services are considered and the QoS attributes are recalculated to see if they fit within the requirements. The concrete workflow is produced and sent back to ArguBroker. ArguBroker translates the workflow and produces the DMML. Figure 6.20 shows the returned workflow. The type conversion shim is located on the same GRIA server as the getImage service, hence the download and upload processes can be skipped.

Figure 6.20: Concrete workflow with shims returned for the pollutionMap scenario

6.7.6 Case Studies Evaluation

The focus of the framework designed in this chapter is supporting the decision making process and not the decisions themselves. The accuracy of the results returns is dependent on the decision-making component, in these particular cases the agent environment, and the only way to evaluate the framework is to determine whether the information provided to the agents is
enough for them to make accurate decisions. The results and workflows returned by the agents show that the information provided is suitable for decision making. Like the abstract implementation, the quality of the information stored in the registry and the terms used to describe the abstract/semantic node play an important role in the resolution process. By introducing ontologies, the terms used to describe the nodes and the descriptions held in the registry are normalised to ensure more accurate matches and this has shown a greater improvement when compared to service selection without ontologies.

6.8 Evaluation

These case studies illustrate how the agent-based Grid-workflow framework extends to support agents who perform decision-making tasks on behalf of the end users. Semantic node definitions resolve to concrete services as expected in simple scenarios. Three categories classify the benefits and resulting disadvantages of this agent-based framework: automation, optimization and flexibility.

6.8.1 Framework Evaluation

The case studies described show how the agent based Grid-workflow framework can be extended to support agents who perform the decision-making tasks on behalf of the end users. Semantic nodes have also been
Moving Towards Automated Service Composition Through the Use of Intelligent Agents

successfully defined and resolved to concrete services as expected in the simple scenarios. The benefits of this agent based framework and the disadvantages that occur as a result can fall into each of the following categories: automation, optimization and flexibility.

**Automation**

The framework supports automatic discovery of services, service composition and reasoning in order to initiate the workflow composition process. The work shown here proves that shimming and workflow design decisions that go beyond individual service matchmaking are subject to automation, but the process still requires manual interaction to ensure the correctness of the final workflow. Increasing the amount of automation that takes place in the workflow composition process also increases the number of points of failure and reliance on the underlying technologies, e.g. if the requestor brokering system fails, the agent environment cannot be contacted.

**Optimization**

The more optimizing variables added to the decision-making process, the more complex and time-consuming the decisions become. Accurate results demand a great deal of preliminary work. Such work ranges over populating the registries and continuously updating the accuracy of their details, refining the ontologies to ensure the terms describe the services available, and
providing the agent system with all the necessities to perform reasoning and handle most eventualities.

The extra architectural components and increased interactions also affect performance. Table 6.2 shows the overheads associated with ArguBroker and its translation tools.

<table>
<thead>
<tr>
<th>Components</th>
<th>Data Complexity</th>
<th>Maximum Response Time (ms)</th>
<th>Minimum Response Time (ms)</th>
<th>Average Response Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArguBroker</td>
<td>Single Node Workflow</td>
<td>473.4</td>
<td>360.1</td>
<td>417.2</td>
</tr>
<tr>
<td>ArguBroker</td>
<td>Ten Node Workflow</td>
<td>533.3</td>
<td>390.1</td>
<td>463.2</td>
</tr>
<tr>
<td>DMML2Prolog</td>
<td>One Node</td>
<td>199.3</td>
<td>160.2</td>
<td>179.7</td>
</tr>
<tr>
<td>Prolog2DMML</td>
<td>One Node</td>
<td>283.5</td>
<td>47.5</td>
<td>154.5</td>
</tr>
<tr>
<td>SLA2Prolog</td>
<td>One Template</td>
<td>282.5</td>
<td>27.4</td>
<td>161.4</td>
</tr>
<tr>
<td>Prolog2SLA</td>
<td>One Template</td>
<td>276.3</td>
<td>31.0</td>
<td>156.8</td>
</tr>
</tbody>
</table>

Table 6.2: Response times for ArguBroker comments

These extra overheads are in milliseconds and thus do not greatly affect decision-making process time.

**Flexibility**

By modularizing the components of the agent based Grid-workflow framework, workflow systems and Grid middleware implementations are
interchangeable, keeping in line with the SOA paradigm. The translation tools handle language incompatibilities so the agent framework and registries implementation may also interchange or support a range of Grid and workflow systems. These translators support different agent implementations and handle more than workflow language. InforSense and Taverna workflows can potentially use ArguBroker to resolve services, for example, as long as the translation tools are there.

While modularity leads to increased implementation flexibility, inevitable tradeoffs distinguish this framework from direct, tightly coupled implementations. For example, language translations may cause loss of functionality or increase authoring time due to the transfer of information between components and the extra processing required.

Addressing Requirements

This chapter outlines four requirements for performing service selection and composition:

- The semantic levels of abstraction address the semantic description requirement, presenting workflow and node annotation for resolution to Grid Services. Note that a combination of different levels can increase result accuracy, but also narrows the search space. The introduction of a semantic handler provides a means to define abstract nodes and
abstract workflows with semantic descriptions. Utilizing ontologies in the framework allows for term normalization and consistency.

- To determine the performance of each individual Grid service, the work presented here defines QoS metrics and fidelity models. Optimizing a workflow requires more than simple matchmaking, thus service composition and workflow models are provided.

- In terms of decision-making, the goal and attribute definitions are sufficient for searching and selecting between services. The composition models aid decision-making by shimming and service locality factors. Registries provide a data store for services and metadata information also required for decision-making.

- Extra interfaces to enable automation include a semantic handler for communicating the abstract requirements to the agent environment, a brokering system and translational tools for addressing the Grid-workflow communication gap, and SLA support to propagate the negotiated QoS service parameters to the workflow author.

### 6.8.2 Service Selection and Composition Models Evaluation

The following graphs and tables show the expected and actual results of the InforSense pollutionMap workflows executing two services on different
GRIA services. For the parallel execution, two branches were executed simultaneously on different GRIA servers, both running getImage and pollutionMap services. For the sequential execution, both branches were arranged to run one after another, i.e. the second branch only ran after the first branch had completed successfully. Table 6.3 and Figure 6.21: Comparisons between sequential, parallel models and actual results show the predicted values for the sequential and parallel executions.

<table>
<thead>
<tr>
<th>Input File Size (Mb)</th>
<th>Seq. Model (Min) (s)</th>
<th>Seq. Model (Max) (s)</th>
<th>Seq. Actual</th>
<th>Par. Model (Min) (s)</th>
<th>Par. Model (Max) (s)</th>
<th>Par. Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>116.3</td>
<td>137.5</td>
<td>122.5</td>
<td>82.7</td>
<td>104.6</td>
<td>81.8</td>
</tr>
<tr>
<td>1</td>
<td>125.9</td>
<td>147.4</td>
<td>134.5</td>
<td>90.0</td>
<td>109.5</td>
<td>104.2</td>
</tr>
<tr>
<td>2</td>
<td>179.8</td>
<td>185.5</td>
<td>176.9</td>
<td>123.9</td>
<td>148.1</td>
<td>128.9</td>
</tr>
<tr>
<td>3</td>
<td>209.6</td>
<td>220.8</td>
<td>213.7</td>
<td>153.6</td>
<td>167.7</td>
<td>148.7</td>
</tr>
<tr>
<td>5</td>
<td>284.7</td>
<td>311.6</td>
<td>287.6</td>
<td>206.9</td>
<td>232.2</td>
<td>214.5</td>
</tr>
</tbody>
</table>

Table 6.3: Comparisons between sequential, parallel composition models and actual results
The models indicate that executing the services in parallel will reduce the execution time compared to executing the services sequentially. While some of the actual values fall out of the model ranges, the general trend lines fall within the expected boundaries.

Figure 6.21: Comparisons between sequential, parallel models and actual results
Like the parallel and sequential results, the trend line for the loop result generally falls within the given boundaries. Increasing the number of experiments helps to refine the models and the error margins.
Moving Towards Automated Service Composition Through the Use of Intelligent Agents

The experiments to generate the models and actual values were performed in a controlled environment. However, in a real life scenario, many other factors must be taken into consideration, such as bandwidth and resource usage. These factors would vary the expected results greatly leading to larger error margins. Furthermore, these experiments consider only two services and relatively small file sizes. Increasing the number of services and input file sizes would also vary the results greatly.

6.9 Summary and Conclusions

The abstract framework extends to support semantic notations for automated Grid service composition and the use of agents for service selection and workflow composition. Specific decisions need to be made when selecting and composing Grid service, and a range of workflow models need to be taken into account. The proposed semantic handler is based on the abstract handler and used in tandem with the Grid handler. In addition to the workflow extensions, a number of components and technologies build up the environment for automated workflow composition.

In terms of workflows, like the abstract handler, the semantic handler needs to be used in conjunction with the Grid handler. The broker implementation must also take into account the design of the Grid handler and nodes required to invoke a service. Separating the broker from the workflow system and the Agent world allows for a range of translators to be added without
affect or being tied to particular implementations. ArguBroker was also designed so that it could be used on both the requestor and provider side.

With regards to Grid middleware, separating the agents, broker, registries and workflow system allows for Grid heterogeneity. Each component in the environment can be extended to support multiple Grid middleware. However, if this is the case, the workflow system must have the relevant Grid handlers installed and the workflow planner in the broker must also be updated to handle the different implementations.
Chapter 7
Summary, Conclusions and Future Work

This chapter summarises the work done with respect to Grid enabling workflows and identifies directions for further work in this field.

7.1 Summary

This thesis outlined a need for a framework that integrates existing workflow systems and Grid middleware from a domain perspective rather than a technological or academic one in accordance with the SOA paradigm. Existing methodologies are either ad-hoc or described at too high a level, revealing a number of challenges associated with defining such a framework. Commonality between implementations of different scientific workflow systems and Grid middleware identified a list of general workflow and Grid components that abstract their common features. Using these components, a generalized layered architecture was defined and the interactions between them identified in order to determine the steps required for executing Grid services via workflow systems. To address accessibility of Grid functionality using workflow nodes, this thesis identified a number of possible workflow node mapping: a single node approach and the splitting of Grid functionality
across nodes utilizing resource allocations or metadata. For each mapping this thesis outlined how the nodes can be implemented, studied the interactions involved in executing a service and presented an evaluation of its usability, and the effect on overall workflow performance. Each set of mappings was evaluated and validated in the context of application scenarios spread across a range of domains, such as Pharmaceutical and Automotive. The work presented here identified that a range of node mappings are required, depending on the Grid functionality available to, and features supported by, a workflow system. The node mappings presented in this thesis have aimed to address different combinations and identified where each would be suitable, determining that a multiple node approach with metadata support is the optimum mapping in that it strikes the best balance between usability, performance and supported Grid functionality.

This thesis also recognises the complexities associated with the authoring of workflows composed of Grid services, and the need for workflow abstraction during the authoring process. The work presented here identifies the criteria for describing user requirements at a high level and supporting these requirements using abstract workflow nodes. Further changes to the proposed framework were included to resolve abstract nodes to concrete service implementations. Identifying a design for abstract Grid nodes led to an updated architecture and studies of the associated component interactions. The updates and interactions were primarily associated with resolving service selection decisions during both authoring and execution.
time. Based on the case studies considered, automated service selection at runtime is preferable to manual selection during workflow authoring. However, automated service selection is generally possible only when services have compatible interfaces.

Given the workflow abstraction and service selection investigation, this thesis then identified how to automate the service composition process at authoring time using autonomous intelligent systems. The components required perform such operations and a revised layered architecture were then defined and the interactions between the different components investigated. Understanding that the success of an agent environment is heavily dependent on the data and information provided to it, led to identifying a number of workflow patterns as well as execution and service resolution models. Noting that it is generally difficult to predict the actual execution of a Grid-based workflow due to the number of runtime variables that affect the performance, the performance models investigated were at a high-level and intended to guide implementation decisions rather than to predict workflow execution time. The framework was also extended to support a brokering system that can be used to bridge the workflow, Grid and agent worlds. The approach is generic and is extensible to support further implementations of each of the technologies, as well as using other decision support tools when developing Grid-based workflows.
7.2 Conclusions

This thesis started by recognised four challenges associated with bringing Grid and workflow technologies together to address the SOA paradigm, forming the basis for the work described herein. The first and second challenges address the simple SOA model consisting of Service Providers and Requestors, while the third and fourth apply to the extended SOA model, which introduces Discovery Agencies. We review the contributions of the work presented in this thesis by revisiting these challenges:

- **How to select and compose Grid services through workflows:**
  Selecting and composing Grid services through workflows effectively required designing a layered architecture to present a global view of the technologies and components involved. In general, workflows must support Grid functionality at the node level, allowing authors to compose Grid services and build further applications. The work presented here has identified that a single form of node mapping is not sufficient given the array of Grid and scientific workflow technologies available. This led to the definition of three forms of node mapping, single, R-DTE and DTE-M. Each mapping has its own advantages and disadvantages in terms of performance, functionality supported, complexity and usability. Interaction diagrams help identify the flow of information across the layers of the architecture for the various forms of node mappings. These interaction diagrams also
indicate whether a particular form of node mapping is simple or complex judging by the volume of components and/or interactions between them. Essentially, all three criteria that make the frameworks presented here are required to support Grid functionality from within workflow systems. The techniques described in this thesis show, within a SOA context, how it is possible to map Grid functionality to workflows in a structured and methodical way. This has been proven through the successful application of the framework to numerous case studies. However, the work presented here has only looked at workflow and Grid technologies for addressing the needs of SOAs. The work presented here did not investigate the applicability of the framework defined to other technologies, such as Cloud computing concepts, which are starting to replace Grid computing concepts in the SOA paradigm.

- **How to address support for heterogeneity:** The Grid-workflow framework presented here successfully supported two forms of heterogeneity and maintained the node mapping definitions described earlier. The Gateway approach maintains a level of user interface consistency, ideal for workflow authors. However, this approach has greater overheads compared to the Adaptive API approach. While the Adaptive API approach optimizes performance through direct access to middleware, the user interface inevitably suffers by having to support interfaces for different middleware. With respect to
heterogeneity, the more middleware supported the greater the compromises in terms of usability for the Adaptive API approach and performance for the Gateway approach.

- **How to provide workflow abstractions for supporting the extended SOA model:** Even though DTE-based node mappings proposed in this thesis have several advantages over a single node mapping, the latter “one node per service” approach is more intuitive when it comes to constructing workflows. Abstraction not only helps to move the DTE-based mappings towards the single node mappings, but also provides the foundations for supporting discovery agencies in the extended SOA model. The work presented in Chapter 5 has successfully shown how framework extensions support Grid-based workflow abstraction using authoring time and runtime service resolution.

**How to utilize intelligent systems in building Grid-based workflows:** An extended framework that supports automated Grid workflow resolution through intelligent systems has been investigated. However, utilizing intelligent systems, such as agents, to support the required decision-making necessitates a brokering system and translation tools to bridge the gap between the different environments. While these intelligent systems can perform the decision-making and negotiating operations on behalf of the workflow author, they still require a large amount of
information and models to be provided from the Service Requestor side. Given the large number of variables involved, these models only act as guides and do not predict the actual outcome of the workflow. In general, this thesis has only identified what types of models and information are required for automating Grid workflows and investigated the components required for exchanging such information with the decision making tools, and did not investigate the quality of the decisions themselves which are beyond the scope of our architecture focus. Furthermore, only simple data flows have been investigated in our work to demonstrate the basic principles, and it remains to be seen whether the methodology of using automated decision-making is viable for complex workflows that use more services and multiple control flow constructs. Furthermore, in order to construct semantic workflows, workflow authors were required to know how to write service descriptions in semantic languages such as WSMO or OWL-S. Ideally, semantic descriptions should be specified in a simpler form using a more basic language or set of terms.

7.3 Future Work

The work described here views the Service Requestor side from a bottom-up approach, i.e. authoring a workflow then publishing it at a higher level for domain experts. With the increasing popularity of web 2.0-based dashboard interfaces [12][48], or mashups which follow a top-down approach, i.e. real-time manipulation of, and interaction with, data from web interfaces (possibly
Summary, Conclusions and Future Work

supported by low-level sub workflows), it would be interesting to see how this would affect the framework and architecture. Fox in [48] provides a comparison between Grids and Web 2.0, identifying their commonalities and differences, even stating that mashups could replace workflows. This increasingly user-centric view of SOAs presents some intriguing challenges with respect to the framework. For example, the interactive nature of mashups means they do not conform strictly to the DAG paradigm and would greatly affect the mapping of Grid functionality to Service Requestor levels of abstraction. Furthermore, the interactions between the components across the layers would change significantly, since the design for the interactions is from a workflow perspective. The focus of these interactions would now be on finding these abstractions for supporting interactive web interfaces.

The work presented in this thesis has only covered a sub-section of the workflow taxonomies and classifications described by Yu in [121]. The described Grid-workflow framework assumes that the workflow system used has a centralized scheduling architecture, opposed to hierarchical or decentralized architectures. In a centralized architecture, one server is responsible for making decisions for all the tasks in a workflow, whereas decentralized and hierarchical allow tasks to be scheduled via sub-workflows. Both [59] and [103] compare these different forms of workflow architecture. It would be interesting to analyze how these different workflow architectures affect the Grid-workflow component interactions and the node mappings, since operations such as data transfer and job invocation would
have to occur across multiple workflow layers. This would notably affect the performance given the extra calls and transfer of data/information between the workflow engines. Furthermore, these extra workflow layers increase the risk of failure, thus reliability becomes an important factor that affects the interactions (choosing the most reliable path) and requires support for reliability QoS constraints, such as those described by Cardoso in [17].

With respect to automating Grid workflows, the work presented here has only considered simple examples and focused solely on building a foundation for further or more complex automation. Complex workflows require more formal methods of interaction, as described by Curcin in [25]. Furthermore, since files are transferred between Grid services, assumptions are made regarding data type matching, thus types are generally ignored. Curcin [25] describes models for resolving data type mismatches and the affect of data quality on workflows. Applying these models to Grid workflows using the type metadata information associated with each service presents a possible research opportunity. The selection and composition models presented in this thesis are based on a historical data approach [121] where previous execution information is used to predict the performance. Yu [121] also suggests simulation and analytical modeling for performance prediction, the former using simulations to emulate the workflow tasks, while the latter uses an analytical metric. Providing support for these performance estimation approaches would also affect the framework, for example, how to simulate Grid workflow executions and pass them to intelligent system for decision-
making, and supporting extra analytical metrics from a workflow system. These extensions may increase the number of required components and interactions that take place in the framework, increasing the complexity.

Overall, the vision is to have agents collaborate with workflow authors and domain experts in creating or adapting workflows with no or little interaction. Yan in [118] lists a number of potential opportunities with respect to agent-enhanced workflows for business processes that are relevant to the work presented here, such as effective communication and negotiation. With respect to communication, agents should be given higher-level specifications unlike the approach presented in Chapter 6, which still requires technical formats and languages such as WSMO. Negotiations have not reached the level of maturity to address real world, complex problems. Such scenarios would require more information from the workflow level and an iterative process for refining workflows, thus changing the interactions between components. This is an ambitious endeavor equivalent to automatic program development. However, the framework presented here has made significant steps to achieving these visions for SOA-based workflow authoring and identifies the key components needed to support this vision.
References


References


[23] C. Comito, C. Mastroianni, and D. Talia, “Metadata, Ontologies, and Information Models For Grid PSE Toolkits Based on Web Services"
References


References


References


References


References


References

Proceedings of the 2009 IEEE/ACM International Joint Conference on
Web Intelligence and Intelligent Agent Technology (WI '09), 15
September 2009, pp. 693-696.


[59] V. Hamscher, U. Schwiegelshohn, A. Steit, and R. Yahyapour,
"Evaluation of Job-Scheduling Strategies for Grid Computing" in 1st
IEEE/ACM International Workshop on Grid Computing (Grid 2000),
Lecture Notes in Computer Science (LNCS), Berlin, Germany, pp 191-

Management Coalition, Hampshire, UK, TC00-1003, Issue 1.1, 19

[61] 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, vol. 34, pp. w729-w732, 1 July
2006.
References


References


References


References


References


References


References


