

ISENGARD: an Infrastructure for Supporting e-Science and Grid Application Development

Donny Kurniawan, BCompSc (Hons)

Submitted in fulfillment of
the requirements for the degree of
Doctor of Philosophy



Caulfield School of Information Technology
Monash University
June, 2009

Declaration

I declare that this thesis contains no material which has previously been submitted for a degree or diploma in any university and, to the best of my knowledge and belief, this thesis contains no material which has previously been published or written by another person, except when due reference is made in the text of the thesis.

Donny Kurniawan

Acknowledgements

This thesis is the culmination of a long journey throughout which I have received great support from many people whom I wish to acknowledge here. This thesis would not have been possible without their support.

First of all, I would like to thank my family, Mum and Sister, for their love and support at all times. They have inspired and encouraged me. This thesis is for them.

I would like to express my gratitude towards my supervisor, David Abramson, for the encouragement, resources, and advice that I have received during my candidature. I am very grateful for his outstanding support that any PhD candidate could possibly hope for.

In addition, I would like to thank Shahaan Ayyub, Blair Bethwaite, Philip Chan, Clement Chu, Ngoc Minh Dinh, Colin Enticott, Slavisa Garic, Rob Gray, Tim Ho, Tom Peachy, and Jefferson Tan for their valuable ideas and encouragement.

I am grateful to the staff of Caulfield School of Information Technology who have kindly helped me during my time at Monash. I acknowledge the support from Michelle Ketchen, Allison Mitchell, Akamon Kunkangkopun, Duke Fonias, and See Ngieng.

I would like to thank my friends ¹ for their support during my candidature.

Finally, my deepest gratitude to my God, for He so loved the world that He gave His one and only Son, that whoever believes in Him shall not perish but have eternal life (John 3:16). *Thank You for what You have done and given.*

¹Colophon

For my Mum and Sister and for my *Lord Jesus Christ*:
the One who plants and waters and causes all things to grow,
to Him in whom all things find their purpose. Soli Deo Gloria.

Abstract

Grid computing facilitates the aggregation and coordination of resources that are distributed across multiple administrative domains. The combined infrastructure provides data storage and processing capacity beyond that of individual organizations; and enables large-scale, complex, and multi-domain e-Science experiments.

Grids are inherently distributed and heterogeneous; and this poses challenges such as data access and authentication methods. To alleviate these concerns, the grid community has developed specific middleware that provides a uniform set of services for security, information, and management. Grid middleware is used to hide the heterogeneous nature and provide users with a homogenous and seamless environment by implementing a set of standardized interfaces to a variety of services.

Writing, deploying, and testing grid applications over highly heterogeneous and distributed resources is complex and challenging. The process requires grid-enabled programming tools that can handle the complexity and scale of the infrastructure. While a large amount of research has been undertaken into grid middleware, little work has been directed specifically at the area of grid application development tools. As a result, there is a lack of tools for building, testing, managing, and deploying grid-enabled software; and this impedes the uptake of grid computing.

This thesis presents the design, implementation, and demonstration of ISENGARD, an infrastructure for supporting e-Science and grid application *development*. ISENGARD consists of three distinct software components with well-defined APIs and bindings. Specifically, they include: an IDE framework which can be incorporated into existing IDEs for developing grid

applications; a middleware-independent framework that provides high-level development services; and a modular framework in the form of a grid debugging service for finding and identifying software bugs in grid applications.

Publications

- Donny Kurniawan and David Abramson, *Worqbench: An Integrated Framework for e-Science Application Development*. In e-Science 2006: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, Amsterdam, Netherlands, December 2006.
- Donny Kurniawan and David Abramson, *A WSRF-Compliant Debugger for Grid Applications*. In IPDPS 2007: Proceedings of the 21th International Parallel and Distributed Processing Symposium, Long Beach, USA, March 2007.
- Donny Kurniawan and David Abramson, *An Integrated Grid Development Environment in Eclipse*. In e-Science 2007: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing, Bangalore, India, December 2007.
- Donny Kurniawan and David Abramson, *An IDE Framework for Grid Application Development*. In Grid 2008: Proceedings of the Ninth International Workshop on Grid Computing, Tsukuba, Japan, September 2008.

Contents

| | |
|---|--------------|
| List of Figures | xix |
| List of Tables | xxi |
| List of Codes | xxiii |
| 1 Introduction | 1 |
| 1.1 Motivation and Objectives | 3 |
| 1.2 Research Contributions | 7 |
| 1.3 Thesis Outline | 8 |
| 2 Background and Literature Review | 9 |
| 2.1 Software Life Cycle | 9 |
| 2.2 Application Development Tools | 11 |
| 2.3 Integrated Development Environments | 12 |
| 2.3.1 Visual Studio | 14 |
| 2.3.2 Xcode | 15 |
| 2.3.3 KDevelop | 16 |
| 2.3.4 Anjuta | 16 |
| 2.3.5 NetBeans | 17 |
| 2.3.6 Eclipse | 18 |
| 2.4 Grid Computing | 20 |
| 2.5 Service-Oriented Grids | 22 |
| 2.5.1 Web Services | 22 |
| 2.5.2 Web Services Resource Framework | 23 |

| | | |
|----------|---|-----------|
| 2.5.3 | Open Grid Services Architecture | 25 |
| 2.6 | Grid Middleware | 26 |
| 2.6.1 | Globus Toolkit | 27 |
| 2.6.2 | UNICORE | 29 |
| 2.6.3 | gLite | 31 |
| 2.7 | Grid Application-Level Tools | 32 |
| 2.8 | Grid Programming Libraries | 33 |
| 2.8.1 | Message Passing Libraries | 33 |
| 2.8.2 | Object-Based and Component-Based Libraries | 34 |
| 2.8.3 | Remote Procedure Call Libraries | 35 |
| 2.9 | Grid Application Execution Environments | 36 |
| 2.9.1 | Grid Portals | 36 |
| 2.9.2 | Workflow Management Systems | 38 |
| 2.9.3 | Parameter Sweep Systems | 39 |
| 2.10 | Grid Integrated Development Environments | 41 |
| 2.10.1 | P-GRADE | 41 |
| 2.10.2 | GT4IDE | 42 |
| 2.10.3 | GriDE | 43 |
| 2.10.4 | g-Eclipse | 43 |
| 2.10.5 | GDT | 44 |
| 2.10.6 | Introduce | 45 |
| 2.11 | Background and Literature Review Summary | 47 |
| 3 | Architecture and Design | 49 |
| 3.1 | Grid Application Development | 50 |
| 3.2 | Architecture of ISENGARD | 52 |
| 3.3 | GridDebug Framework | 54 |
| 3.3.1 | GridDebug Components | 55 |
| 3.3.1.1 | Debug Client | 55 |
| 3.3.1.2 | Middleware Compatibility Layer | 56 |
| 3.3.1.3 | Debug API Library | 56 |
| 3.3.1.4 | Debug Back-End | 57 |
| 3.3.2 | GridDebug Application Programming Interface | 57 |

| | | |
|----------|---|-----------|
| 3.3.3 | Grid Remote Debugging | 59 |
| 3.3.4 | GridDebug Design Conclusion | 61 |
| 3.4 | Worqbench | 62 |
| 3.4.1 | Worqbench Components | 63 |
| 3.4.1.1 | Connection Interface | 64 |
| 3.4.1.2 | Code Repository | 64 |
| 3.4.1.3 | Development Manager | 66 |
| 3.4.1.4 | Database | 66 |
| 3.4.1.5 | Development Services | 66 |
| 3.4.1.6 | Middleware Adapter | 67 |
| 3.4.2 | System Model | 68 |
| 3.4.3 | Worqbench Design Conclusion | 71 |
| 3.5 | Remote Development Tools | 72 |
| 3.5.1 | RDT Framework | 73 |
| 3.5.1.1 | Entity Manager | 75 |
| 3.5.1.2 | Metadata Manager | 75 |
| 3.5.1.3 | Event Manager | 76 |
| 3.5.1.4 | RDT Application Programming Interface | 76 |
| 3.5.2 | RDT User Interface | 77 |
| 3.5.2.1 | RDT Entity Viewer | 77 |
| 3.5.2.2 | Remote Launching Support | 77 |
| 3.5.2.3 | Multi-View Code Editor | 78 |
| 3.5.3 | RDT Design Conclusion | 81 |
| 3.6 | Architecture and Design Summary | 82 |
| 4 | System Implementation | 85 |
| 4.1 | GridDebug Implementation | 85 |
| 4.1.1 | Middleware Service | 86 |
| 4.1.2 | GridDebug Library and API | 90 |
| 4.1.2.1 | Debugger Initialization and Termination | 92 |
| 4.1.2.2 | Process Sets | 92 |
| 4.1.2.3 | Actionpoints | 93 |
| 4.1.2.4 | Execution Control | 93 |

| | | |
|----------|--|------------|
| 4.1.2.5 | Program Information | 93 |
| 4.1.2.6 | Data Display and Manipulation | 94 |
| 4.1.3 | GridDebug Back-End | 94 |
| 4.2 | Worqbench Implementation | 98 |
| 4.2.1 | Client/Server Arrangements | 100 |
| 4.2.2 | Client Connectors | 100 |
| 4.2.3 | Grid Middleware Adapters | 103 |
| 4.2.4 | Worqbench Data Structures and APIs | 105 |
| 4.2.5 | Worqbench Services | 108 |
| 4.2.5.1 | Packaging Service API | 109 |
| 4.2.5.2 | Launching Service API | 109 |
| 4.2.5.3 | Editing Service API | 110 |
| 4.3 | RDT Implementation | 111 |
| 4.3.1 | IDE Integration | 112 |
| 4.3.2 | User Interface | 114 |
| 4.3.3 | RDT Data Structures | 114 |
| 4.3.4 | RDT Framework API | 116 |
| 4.3.4.1 | Event Listener and Callback | 117 |
| 4.3.5 | RDT Multi-View Code Editor | 118 |
| 4.4 | System Implementation Summary | 119 |
| 5 | System Demonstration | 121 |
| 5.1 | Grid Testbed and Software | 123 |
| 5.2 | Case Study 1: Client Access and Management | 126 |
| 5.2.1 | Client Access | 127 |
| 5.2.2 | Entity Management | 131 |
| 5.3 | Case Study 2: IDE Scripting and Code Editing | 136 |
| 5.3.1 | Automation Scripts | 137 |
| 5.3.2 | Event Scripts | 140 |
| 5.3.3 | Code Editing with Multiple Views | 144 |
| 5.4 | Case Study 3: Software Execution and Debugging | 149 |
| 5.4.1 | Software Execution | 150 |
| 5.4.2 | Debugging Grid Applications | 157 |

| | | |
|----------|---|------------|
| 5.5 | Case Study 4: Molecular Dynamics Simulation | 160 |
| 5.6 | System Demonstration Summary | 165 |
| 6 | Future Directions and Conclusions | 167 |
| 6.1 | Thesis Summary | 168 |
| 6.2 | Future Research Directions | 169 |
| 6.2.1 | Advanced Development Support in Grid Middleware . | 169 |
| 6.2.2 | Innovative Grid Development Environments | 170 |
| 6.2.3 | Development Tools for High-Level Grid Applications . | 171 |
| 6.3 | Thesis Conclusions | 171 |
| A | GridDebug API Methods | 175 |
| B | Worqbench API Methods | 177 |
| C | Remote Development Tools API Methods | 181 |
| D | GridDebug Debug Clients | 185 |
| | Bibliography | 223 |

List of Figures

| | | |
|------|--|-----|
| 2.1 | Web service and WS resources relationship | 24 |
| 3.1 | Architecture of ISENGARD | 53 |
| 3.2 | GridDebug architecture | 56 |
| 3.3 | Components of the debug API library | 58 |
| 3.4 | Grid remote debugging | 61 |
| 3.5 | Three-tier relationship model | 63 |
| 3.6 | Key components in Worqbench | 65 |
| 3.7 | System model of Worqbench | 69 |
| 3.8 | RDT architecture overview | 73 |
| 3.9 | Interactions between development and launching machines . . | 78 |
| 3.10 | Standard and multi-view code editors | 80 |
| 3.11 | Design of multi-view code editor | 81 |
| 4.1 | GridDebug implementation overview | 86 |
| 4.2 | Interaction between GridDebug client and server components . | 87 |
| 4.3 | Components of GridDebug service implementation | 88 |
| 4.4 | GridDebug library components | 91 |
| 4.5 | Class diagram for GDBDebugger | 94 |
| 4.6 | Interaction between GDB and GDBServer | 96 |
| 4.7 | GridDebug debugging session | 97 |
| 4.8 | Worqbench client/server arrangements | 101 |
| 4.9 | Client connectors | 102 |
| 4.10 | Middleware adapters | 104 |
| 4.11 | Interaction between Worqbench, CLI tools, and GT4 services . | 104 |
| 4.12 | Worqbench model classes | 106 |

| | | |
|------|--|-----|
| 4.13 | Eclipse RDT and Worqbench | 113 |
| 4.14 | Class diagram for RDT models | 115 |
| 5.1 | Network layout of the testbed | 123 |
| 5.2 | Topology of the testbed clusters | 125 |
| 5.3 | Worqbench login page | 127 |
| 5.4 | Management page for system resources | 128 |
| 5.5 | Management page for user resources | 128 |
| 5.6 | Eclipse RDT and its Remote Development perspective | 129 |
| 5.7 | Eclipse preferences window for Worqbench | 130 |
| 5.8 | Accessing Worqbench from an IRB session | 131 |
| 5.9 | Execution of a script to initialize access to user resources | 132 |
| 5.10 | Management page for user resource sets | 134 |
| 5.11 | Management page for user projects | 134 |
| 5.12 | Remote Projects and Remote Resources views | 135 |
| 5.13 | Management page for user sessions | 136 |
| 5.14 | Groovy Shell view | 138 |
| 5.15 | Navigator and Groovy Shell views | 139 |
| 5.16 | Eclipse preferences window for Event Callbacks | 144 |
| 5.17 | Multi-view code editor (normal view) | 145 |
| 5.18 | Eclipse preferences window for Multi-View | 146 |
| 5.19 | New Code View dialog boxes | 146 |
| 5.20 | Multi-view code editor (sub-view example 1) | 147 |
| 5.21 | Multi-view code editor (sub-view example 2) | 147 |
| 5.22 | Code editing (sub-view example 1) | 148 |
| 5.23 | Code editing (normal view) | 149 |
| 5.24 | Eclipse launch configuration dialog box (run) | 151 |
| 5.25 | Session Configuration dialog box | 151 |
| 5.26 | Execution of the Jacobi application | 152 |
| 5.27 | Remote Tasks view | 153 |
| 5.28 | Management page for user tasks | 153 |
| 5.29 | Using the Worqbench API to run grid software (1) | 155 |
| 5.30 | Using the Worqbench API to run grid software (2) | 156 |

| | | |
|------|---|-----|
| 5.31 | Eclipse launch configuration dialog box (debug) | 157 |
| 5.32 | Eclipse Debug perspective (1) | 158 |
| 5.33 | Eclipse Debug perspective (2) | 159 |
| 5.34 | Topology of the debugging demonstration | 159 |
| 5.35 | Developing GROMACS in Eclipse | 162 |
| 5.36 | GROMACS development session | 162 |
| 5.37 | GROMACS source code in two different editors | 163 |
| 5.38 | Testing the GROMACS package | 164 |

List of Tables

| | | |
|-----|--|-----|
| 4.1 | Java packages that implement Eclipse RDT | 112 |
| 4.2 | Eclipse RDT events | 118 |
| 5.1 | Testbed resources | 124 |
| A.1 | Debugger initialization and termination methods | 175 |
| A.2 | Methods associated with process sets | 175 |
| A.3 | Data display and manipulation methods | 175 |
| A.4 | Methods associated with actionpoints | 176 |
| A.5 | Execution control methods | 176 |
| A.6 | Methods associated with program information | 176 |
| B.1 | System resource and user API methods | 177 |
| B.2 | UserResource API methods | 178 |
| B.3 | UserProject API methods | 178 |
| B.4 | UserResourceSet API methods | 179 |
| B.5 | UserSession API methods | 179 |
| B.6 | UserTask API methods | 180 |
| C.1 | Methods associated with RDT entities (IResource, ISet) . . . | 181 |
| C.2 | Methods associated with RDT entities (IProject, ISession, ITask) | 182 |
| C.3 | Methods associated with RDT entity managers | 183 |

List of Codes

| | | |
|-----|--|-----|
| 4.1 | A Python script to upload a project zip file | 103 |
| 4.2 | An example of a UserSession's configuration | 108 |
| 4.3 | A Groovy script that utilizes RDT API | 117 |
| 5.1 | A Python script to initialize access to user resources | 133 |
| 5.2 | A Groovy script to list software projects | 138 |
| 5.3 | A Groovy script to create resource sets based on projects . . . | 141 |
| 5.4 | Examples of event callback methods | 143 |
| D.1 | A command-line interface (CLI) GridDebug debugger | 185 |
| D.2 | A Jython script that performs a comparison between variables | 189 |

Chapter 1

Introduction

The Large Hadron Collider (LHC) at CERN laboratory in Geneva is the world's most powerful particle accelerator [LHC, 2007]. It will be used by scientists to answer fundamental questions in particle physics. The research undertaken at LHC will generate massive amounts of data which is predicted to be equivalent to more than 20 million CDs (15 petabytes) per year [LCG, 2007]. It is anticipated that analyzing and processing the resultant data from LHC will take the equivalent of 100,000 of today's personal computers [LHC Communication, 2007]. To tackle these computing needs, CERN has prepared the vast computing infrastructure required by employing grid computing services which are spread across research institutions and universities in Europe, America, and Asia [Messina, 2004, LCG, 2007].

The Sloan Digital Sky Survey (SDSS) is the largest astrophysical study ever undertaken [SDSS, 2007]. It aims to obtain observations on around 100 million astronomical objects and address important questions about the nature of the universe. SDSS is a collaborative project utilizing a dedicated optical telescope, image processing units, and data storage in 25 institutions around the world. The telescope delivers around 200 gigabytes of raw data per night and SDSS has produced 40 terabytes of data in total since it started in 2001 [Xiang et al., 2006, SDSS, 2007].

LHC and SDSS share one distinctive feature: the cooperation between several organizations for common objectives. The scale, complexity, and

scope of the research require capacity beyond that of individual organizations. LHC and SDSS are examples of e-Science research. e-Science is defined as science increasingly done through distributed global collaborations enabled by the Internet, using very large data collections, terascale computing resources, and high performance visualization [Hey and Trefethen, 2003, NeSC, 2007]. It has much to benefit from grid computing which is a special type of distributed computing on a world-wide scale.

Grid computing facilitates the aggregation of resources which are distributed across multiple administrative domains to act as a single large system [Foster et al., 2001]. It presents a paradigm similar to an electric power grid where a variety of resources such as powerful computers, scientific instruments, and data servers contribute their utility into a shared pool for users to access. Much grid research focuses on developing middleware infrastructure in the domains of security, information, and management. This results in a set of standard tools and practices, for example, a single sign-on authentication framework, utilities to manage resources, monitoring services, and a common mechanism to execute grid software. Utilizing grid resources requires appropriate software which needs to be specifically written to take advantage of the infrastructure. However, no common protocols and standards have been defined in the area of grid application development [Kurniawan and Abramson, 2006]. There is a lack of tools for building, testing, managing, and deploying grid-enabled software. For example, there is no standard technique for debugging grid applications across multiple heterogeneous resources [Kurniawan and Abramson, 2007a].

The aim of this thesis is to present the design and implementation of an infrastructure for supporting the development; testing and debugging; and deployment of e-Science applications. The infrastructure, ISENGARD, provides a set of APIs, services, and tools that simplify the task of grid programmers and e-Scientists. ISENGARD consists of three distinct software components with well-defined APIs and bindings [Kurniawan and Abramson, 2006, 2007a,b, 2008]. These components include user-level applications, high-level tools, and low-level middleware services.

The remainder of this chapter is organized as follows: Section 1.1 discusses

the motivation for the research work and the objectives that need to be met. Section 1.2 presents the research contributions of the thesis and Section 1.3 presents the thesis outline.

1.1 Motivation and Objectives

Grid middleware hides much of the complexity of the underlying resources and network fabric. However, in spite of the significant advances in middleware, creating grid applications is still difficult and error prone. Consider the following quotes from a user survey report in 2004 [Balle and Hood, 2004]:

”Unfortunately we are not able ... to understand why so few people are running or developing programs for the grid. Our guess is that developing applications for the grid is too hard and too time consuming. By surveying the landscape for grid-aware tools, we realize that few robust tools are currently available.”

”From a debugging standpoint, most developers are using *standard* debugging techniques to debug applications on large scale systems as well as on the grid. These techniques and tools are adequate in a *non-grid* environment.”

”If we want users to migrate ... to the grid, it is important that we make the tools they need and are accustomed to available on the grid.”

Grid application development is different from the traditional counterpart. The process of writing, deploying, and testing grid applications over highly heterogeneous and distributed resources is complex and challenging. The process requires grid-enabled programming tools that can handle the complexity and scale of the infrastructure. The lack of such programming tools has impeded the uptake of grid computing to date. A study of grid users conducted in 2007 by the UK e-Science Core Programme also confirms the aforementioned issues [Newhouse et al., 2007].

Currently, e-Scientists and grid developers typically rely on disintegrated CLI (command-line interface) tools. Specifically, programmers write applications on a local machine and then transfer them to grid nodes for execution. Debugging is usually performed by logging into the execution nodes, using a range of techniques from displaying program state with `print` statements through to running a command-line debugger such as GDB. Applications also need to be compiled and built according to the platform of the targeted nodes, and this again requires a user to log in to build the software. These software development techniques are time consuming, error prone, inefficient, and do not scale to large grid testbeds [Balle and Hood, 2004].

There are several specific challenges that make developing grid applications non-trivial. The challenges are listed as follows:

1. Scalability with a large number of grid resources.

It is possible to manually build, deploy, and debug a grid application over two or several resources. However, as the number of grid resources being used escalates, the complexity and the manageability factors in using these resources also increase. With the possibility of a large number of compute nodes being used to run grid software, the ability of the development process to scale becomes an important consideration. The current approach of opening multiple terminal windows and employing scripts to semi-automate the process is labor intensive, error prone, and ineffective. The situation may also be exacerbated if a programmer wishes to use resources which are handled by two or more distinct grid middleware with different specifics, such as login procedures and user environments.

2. Heterogeneity in the grid environment.

Although grid middleware defines a single set of common services for security, information, and management; and offers a uniform way for executing grid applications, it does not completely solve the problem of heterogeneity. In the grid environment, resources with different architectures and operating systems can be mixed and used collaboratively.

These differences must be taken into account by programmers since grid applications still need to be written and compiled in accordance with the underlying execution hardware.

The traditional way of handling heterogeneity in the application development domain is to employ abstraction to hide the underlying differences and expose higher-level functions. For example, writing the applications in high-level languages that have portable interpreters and virtual machines (such as Python and Java). However, this approach has several drawbacks. It is not suitable for performance-critical programs and it may not fully exploit the advantages offered by some specific architectures. Moreover, in practice, the majority of grid programmers still write the applications in system languages such as C/C++ or Fortran [Balle and Hood, 2004].

3. Lack of standards in the domain of grid application development.

Standards and protocols have been proposed and defined in grid computing for the execution management of grid applications. However, no standard has been defined in the domain of grid application development. For example, there is no common protocol or technique for software development features such as compiling and debugging grid applications. This lack of standard can lead to re-implementation of development tools that may not be compatible or interoperable and may only work for certain grid middleware. Alternatively, the tools and services need to be designed in a modular way and work with a wide variety of grid middleware.

4. Ease of use of grid programming tools

Programmers accustomed to traditional application development may find a high barrier to entry with grid programming because of the additional considerations such as scalability and heterogeneity involving multiple resources. Over and above addressing these peculiar issues, grid programming tools must also be user-friendly and easy to use. We believe that it is advantageous to have grid programming tools with

a familiar and intuitive user interface to the traditional application developers and e-Scientists.

There is a need to have grid-enabled programming infrastructure and tools that can support e-Scientists in developing grid applications effectively and with ease. The lack of such tools has encumbered e-Scientists from collaborating and utilizing grid infrastructure for large-scale scientific research [Balle and Hood, 2004].

In contrast, in the traditional non-grid environment, developers have access to a range of powerful development tools and platforms. For example, integrated development environments (IDEs) combine various programming tools into one cohesive environment. Although IDEs have a steep learning curve and can be difficult to master [Seffah and Rilling, 2001, Reis and Cartwright, 2004, Kline and Seffah, 2005], various studies have shown that once a particular environment has been mastered, it can significantly increase programmers' productivity [Case, 1985, Norman and Nunamaker, 1989, Tsuda et al., 1992, Glass, 1999, Zeller, 2007]. e-Scientists and grid developers have much to gain from using IDEs. However, almost all current platforms are geared towards traditional application development on a local machine, and thus do not address many of the peculiar aspects of grid software development that make it difficult.

To fill this gap and to meet the challenges listed in this section, we have designed and implemented ISENGARD, an infrastructure for supporting e-Science and grid application development. It presents several novel contributions towards improving the development of grid-based software. The infrastructure consists of three software components, namely Remote Development Tools (RDT) [Kurniawan and Abramson, 2007b, 2008], Worqbench [Kurniawan and Abramson, 2006], and GridDebug [Kurniawan and Abramson, 2007a].

RDT is an IDE framework for e-Science and grid application development. It can be incorporated into a number of existing integrated development environments, further leveraging the advantages of such systems. It exposes resource heterogeneity in a controlled manner in an IDE and provides a

tight integration between the execution platforms and the code development process.

Worqbench is a framework that provides high-level services and API that can be used by clients or IDEs to package, launch, and debug grid applications. It allows the aggregation of grid resources with different middleware and architectures into a single grid resource set.

GridDebug is a debugging framework in the form of a grid service for debugging and testing grid applications. It provides a debug library with a well-defined application programming interface. The API specifies a generic debug model and a set of methods and objects for grid application debugging.

1.2 Research Contributions

The main contribution of this thesis is the design and implementation of a software infrastructure for grid application development, ISENGARD. It introduces a number of innovative features which are listed as follows:

1. The design of a grid service debug architecture suitable for testing and debugging computational grid applications. The service is modular and it can be incorporated into any existing grid middleware. The debugging service is generic and it can utilize any back-end debuggers to perform the low-level debugging operations.
2. The specification and design of an application programming interface for grid debugging. The API is used by the grid service debug architecture and it can also be used within a grid level debugger or other high level software tools that require these functions. The API defines a generic debug model and a collection of objects and methods for grid application debugging. The API includes, for example, methods to define process sets, to create/delete breakpoints, and to control the execution of programs.
3. A modular and extensible framework architecture that manages the interaction between grid middleware and IDEs. The framework has been

designed in a generic way and it is independent of both the middleware and the IDEs. It allows developers to utilize a collection of grid resources where each resource is managed by different grid middleware.

4. The specification of a formal and comprehensive resource and development model for computational grid applications. The model formalizes entities such as application projects, grid resources, resource sets, and development sessions.
5. A framework and a set of tools for integrated development environments that promote a tight integration between the software development process and the execution platforms. It exposes resource heterogeneity in a controlled manner and it regards grid resources and resource sets as first-class objects similar to files, folders, and projects in an integrated development environment.
6. The design of a novel multi-view code editor that separates the display of source code from the textual representation. The editor aids programmers in comprehending the source code since it can show parts of the code that are relevant to a certain context and hide the irrelevant parts.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 provides the background information and literature review on grid application development. Chapter 3 presents the architecture and design of ISENGARD followed by a detailed description of the implementation in Chapter 4. Chapter 5 presents the system demonstration of the research work. Future directions and conclusions are discussed in Chapter 6.

Chapter 2

Background and Literature Review

This chapter presents background information and literature review on grid application development. It discusses various stages of software development and surveys programming tools for traditional and grid-enabled applications. It presents an introduction to grid computing, service-oriented grids, middleware, and high-level programming tools.

The chapter sections are as follows: Section 2.1 gives a discussion on software life cycle. Sections 2.2 and 2.3 survey various application development tools and integrated development environments. An introduction to grid computing and service-oriented grids is presented in Sections 2.4 and 2.5 followed by a survey of grid middleware in Section 2.6. Section 2.7 discusses grid application level tools which can be categorized into three: grid programming libraries (Section 2.8), grid application execution environments (Section 2.9), and grid development environments (Section 2.10). Finally, Section 2.11 gives a summary of this chapter.

2.1 Software Life Cycle

All software undergoes a series of production phases, such as design, implementation, and testing. The study that is concerned with all aspects of

software production is called software engineering [Sommerville, 2006], which can also be defined as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [IEEE, 1990]. The best-known and oldest software development process is the waterfall model [Royce, 1970, Sommerville, 2006]. In this model, developers follow a number of principal phases in order. The phases are: requirements definition; system and software design; implementation and unit testing; integration and system testing; and operation and maintenance. These phases form a software life cycle.

The research work presented in this thesis is concerned with the implementation and unit testing phase. Specifically, the phase is composed of several engineering stages: editing, building, and launching. These stages, which may be repeated several times, form a distinct and smaller software development life cycle. They are described as follows:

- **Editing stage.** In this stage, a developer implements software based on the design specification by writing source code and scripts in some programming languages. The developer may utilize code editors or high-level programming tools such as visual workflow editors. The developer may also link the code with software libraries and toolkits that provide specialized and higher-level functions.
- **Building stage.** In this stage, development tools such as compilers and interpreters are used to translate source code files into a program that can be launched or executed on the intended platforms. If required, additional libraries and modules are also linked into the executable. For software implemented in a compiled programming language, the building stage is typically performed on the execution platforms, unless the developer utilizes cross compilers. If the developer writes the software in an interpreted language, this stage may simply involve the transfer of scripts to the execution platforms.
- **Launching stage.** In this stage, the program is launched or executed to perform its intended function. There are two types of program launchings: normal mode and debug mode. In the normal mode, the program

is executed from start to finish, provided that it does not have any errors. In the debug mode, the program is stopped in the middle of execution and it can be inspected for potential errors. If an error is found, the developer can rectify it by editing the source code.

Grid-based software follows the same life cycle, although there are significant differences in the details of the stages that make traditional development tools inappropriate [Abramson, 2006]. For example, application deployment over heterogeneous resources is rarely supported by traditional tools.

2.2 Application Development Tools

The lowest-level abstraction of an application is machine code which can be defined as a set of processor instructions that is directly executed by a computer. One level higher than machine code is assembly language. It consists of human-readable processor directives or mnemonics for a specific architecture. A program written in assembly is translated into machine code by an assembler. Machine code and assembly language are concerned with low-level details such as machine registers, data bytes, and call stacks. To simplify software development, high-level programming languages were conceived to provide abstractions such as variables, strings, integers, file input/output, and so forth.

Based on the execution model, high-level programming languages can be categorized into three types: compiled, interpreted, and hybrid [Watt and Brown, 2000, Scott, 2006]. Source code written in a compiled language is processed and transformed into a running program by a compiler. This program can then be executed by the computer. Code written in an interpreted language is processed and executed directly by an interpreter. In the hybrid mode, code is compiled into an intermediate representation or bytecode which can then be executed by an interpreter or a virtual machine.

In addition to compilers and interpreters, there are other programming tools that assist developers in creating innovative and bug-free software in straightforward, methodical, and efficient manner. A broad list of application

development tools with notable examples is given below.

- Compilers: GCC [GCC, 2007], Intel C++ Compiler [ICC, 2008].
- Interpreters: AWK [Aho et al., 1988], Python [Python, 2008], Perl [Perl, 2008], Ruby [Ruby, 2008].
- Debuggers: GDB [GDB, 2007], Intel Debugger [IDB, 2008], Valgrind [Valgrind, 2008].
- Profilers: TAU [TAU, 2008], DTrace [DTrace, 2008], GNU gprof [Binutils, 2008].
- Build tools: Make [Make, 2006], Ant [Ant, 2007], Rake [Rake, 2006].
- Revision control systems: CVS [CVS, 2006], Subversion [Subversion, 2007], Bazaar [Bazaar, 2008].
- Parser generators: Yacc [Johnson, 1979], Bison [Bison, 2008], ANTLR [ANTLR, 2008].
- Documentation generators: Doxygen [van Heesch, 2008], Javadoc [Javadoc, 2008].
- Text editors: Vim [Vim, 2008], Emacs [Emacs, 2008], TextMate [MacroMates, 2008].

2.3 Integrated Development Environments

Developing an application and supporting the full software development life cycle require a variety of programming tools. A closer integration of these tools has been shown to expedite and streamline the development process [Brown and McDermid, 1992]. This observation leads to the creation of an integrated development environment (IDE). The idea of an IDE is not particularly new, one of the earliest IDEs was built for the BASIC programming

language in 1963 [Kemeny and Kurtz, 1964, Boekhoudt, 2003]. It was a text-based IDE that seamlessly integrates code editing, compilation, and program execution [Kemeny and Kurtz, 1964, Kurtz, 1981].

Today, an integrated development environment (IDE) typically combines various programming tools such as a source code editor, a compiler, and a debugger into one cohesive environment. This integration allows the tools to work together seamlessly, for example, the debugger can highlight source code errors in the editor and the compiler can be invoked automatically when the code changes. An IDE is a type of CASE tool [Fuggetta, 1993]. CASE, which stands for computer-aided software engineering, encompasses a wide range of tools and methods that are used to support software engineering activities such as requirement analysis, system modeling, and testing [Somerville, 2006]. A thorough classification of CASE technology is given by Fuggetta [1993]. The term IDE is synonymous with integrated project support environment, software engineering environment, software development environment, and CASE workbench.

IDEs which provide various automation and advanced features have a steep learning curve and can be difficult to master [Seffah and Rilling, 2001, Reis and Cartwright, 2004, Kline and Seffah, 2005]. However, various studies have shown that once a particular development environment has been mastered, it can significantly increase programmers' productivity [Case, 1985, Norman and Nunamaker, 1989, Tsuda et al., 1992, Glass, 1999, Zeller, 2007]. IDEs are effective because they improve productivity in three areas, namely efficiency, management control, and quality [Case, 1985, Zeller, 2007]:

- **Efficiency.** IDEs provide various automation functions, such as refactoring and code-generation support, that can substantially reduce the amount of time required to develop an application. IDEs also promote efficiency by providing development tools that have consistent and cohesive user interfaces.
- **Management Control.** IDEs equip programmers with better direct and automated control over the development process through features such as automatic recompilation and source code tracking in the debugging

phase. In addition, tools such as class browsers and method flow tracers simplify the programmers' task in comprehending the source code.

- **Quality.** IDEs improve code quality by providing tools such as an automatic syntax checker, an integrated help system, and a source code documentation generator. IDE utilities such as unit testers, profilers, and program analyzers assist programmers in detecting code defects at the early stage of development.

The following sections describe a number of notable IDEs: Visual Studio (Section 2.3.1) which is the dominant IDE on the Microsoft Windows platform; Xcode (Section 2.3.2) which is included by default in Mac OS X; KDevelop (Section 2.3.3) and Anjuta (Section 2.3.4) which are IDEs for the KDE and GNOME desktop environments respectively; NetBeans (Section 2.3.5) which is an IDE from Sun Microsystems; and Eclipse (Section 2.3.6) which is an advanced and extensible IDE from the Eclipse Foundation.

2.3.1 Visual Studio

Visual Studio is a suite of application development tools from Microsoft for the Windows platforms on servers, workstations, personal digital assistants (PDAs), and smartphones [Visual Studio, 2008]. Visual Studio can also be used to create applications which target Microsoft Office, the .NET framework, and ASP.NET web framework. It supports several programming languages such as C, C++, C#, J#, ASP.NET, and Visual Basic .NET. Additional third party extensions are available to support other languages. Visual Studio includes various Windows software development kits (SDKs) and it utilizes Microsoft compilers and debuggers.

Microsoft provides Visual Studio Team System [VSTS, 2008] for cooperation between developers in a software team. It is designed as a solution for project managers, programmers, and testers to collaborate and communicate more effectively in developing Windows-based applications. It consists of a server, Team Foundation Server, and a suite of client-side tools. Together

they provide functionalities such as project management, version control, build management, work item tracking, metric analysis, and reporting tools.

Microsoft also provides the Visual Studio SDK for Visual Studio Industry Partner (VSIP) members that can be used to extend Visual Studio with additional functions [VSIP, 2008]. The core foundation of the SDK is the Visual Studio Shell that provides a base integrated development environment that can host custom tools and programming languages. The Shell comes in two modes of operation:

- Integrated mode. Applications built on the integrated Shell will be hosted in the Visual Studio IDE. The mode is used if a vendor wants to provide additional functions in the IDE, for example, support for new programming languages.
- Isolated mode. Applications built with the isolated Shell will be unique standalone programs. The mode is used if a vendor wants to develop a specialized tool that does not interact with the Visual Studio IDE but still leverages the Shell's features.

2.3.2 Xcode

Xcode Tools is a suite of development tools from Apple for developing software on the Mac OS X platform [Xcode, 2008]. The suite includes an integrated development environment (Xcode IDE), a tool for designing applications' interfaces (Interface Builder), a performance analyzer tool (Instruments), a widget development tool (Dashcode), and a set of programming libraries and interfaces.

The IDE is the main application of the suite and it uses a modified version of the GNU Compiler Collection (GCC) [GCC, 2007] and GDB [GDB, 2007] as the back-end debugger. It supports developing applications in Java, C, C++, and Objective-C with a variety of application programming interfaces such as Cocoa [Cocoa, 2008] and Carbon [Carbon, 2008]. Interface Builder is an Xcode program for designing and testing user interfaces for Cocoa and Carbon applications. Its graphical editor can be used by developers

to create interfaces that follow Mac OS X human-interface guidelines. The Instruments program is a visual analysis tool for software performance. It allows developers to fully measure, evaluate, and inspect applications running on Mac OS X. It utilizes the DTrace tracing framework in the operating system kernel (Mac OS X 10.5).

Xcode can be extended and automated by creating AppleScript plug-ins [AppleScript, 2008] and Automator workflows [Automator, 2008]. Xcode supports distributed compilation through the use of distcc [distcc, 2007]. It speeds up the compilation of source code by spreading the task across several machines on a network.

2.3.3 KDevelop

KDevelop is a software development product for the K Desktop Environment (KDE) [KDevelop, 2008]. It is implemented in C++ using the Qt GUI toolkit [Qt, 2008] and it is specifically suited for writing KDE and Qt applications. It supports a large number of programming languages such as C, C++, Fortran, Java, Perl, Python, and Ruby. KDevelop features include a class browser, a GUI designer, front-ends for GNU Compiler Collection [GCC, 2007] and GNU Debugger [GDB, 2007], application wizards, a cross-compiling support, a documentation generator, and a revision control support.

KDevelop has a modular architecture with plug-ins and KDE's component framework that allows the IDE to utilize KDE applications such as Kate (KDE Advanced Text Editor) [Kate, 2008]. A number of KDevelop plug-ins exist to provide support for MPI programming, code bookmarks, source formatters, and regular expressions.

2.3.4 Anjuta

Anjuta is an integrated development environment for the GNOME desktop environment [Anjuta, 2008]. It features a number of programming tools that include a project manager, an interactive debugger based on GDB [GDB, 2007], an integrated Glade UI designer [Glade, 2008], a performance profiler, and a source code editor with syntax highlighting and code completion.

Glade is a development tool to create user interfaces for GTK+ [GTK+, 2008] and GNOME [GNOME, 2008] applications. Anjuta supports C and C++ programming languages.

Anjuta has an extensible plug-in framework and almost all IDE features such as the code editor, the project manager, and the debugger back-end are implemented as plug-ins. The plug-ins can be dynamically enabled or disabled and they allow developers to customise the programming environment as required.

2.3.5 NetBeans

NetBeans from Sun Microsystems refers to two distinct products: a platform and an IDE. The NetBeans Platform is a software framework for building cross-platform Java desktop applications [NetBeans, 2007a]. It facilitates the development of rich client applications by providing a number of advanced features such as:

- User interface management. The Platform provides high-level presentation components, for example windows, menus, toolbars, editors, palettes, and wizards. It results in a standard and consistent user interface for rich client applications.
- Data and presentation management. The NetBeans Platform contains a rich toolset for data manipulation and presentation.
- Setting management. A simple and transparent mechanism for saving and restoring user settings and preferences is provided by the Platform.
- Storage management. The Platform offers an abstraction of file-based data access. It provides a common set of APIs to access files regardless of where they are stored. A file object in the NetBeans paradigm may be physically stored on an FTP server, a local disk, or in a database.
- Additional components. The NetBeans Platform has been designed in a modular way. Thus, it is possible to extend the Platform with additional functionalities required by the rich client applications. Available

extensions include specialized editors, remote data access, and automatic network update support.

The NetBeans Platform provides these high-level management functions to allow developers to concentrate on implementing complex and functional applications without being concerned with low-level user interface details.

The NetBeans IDE is an integrated development environment written in Java and built on top of the NetBeans Platform [NetBeans, 2007b]. It can be used to create desktop, web, mobile, and NetBeans-based rich client applications. It has a module-based architecture and the IDE can be extended with various packs. A NetBeans IDE pack is a collection of related modules to support a new feature. Some examples of the packs are: C/C++ Pack for developing C/C++ applications, Enterprise Pack for writing service-oriented architecture (SOA) applications, and Ruby Pack for programming in the Ruby language. Other notable IDE features are:

- A NetBeans Swing GUI builder for developing Java applications with the Swing toolkit [Fowler, 2007].
- Project and build systems based on Apache Ant [Ant, 2007].
- An integrated version control which supports CVS and Subversion [Subversion, 2007].
- An extensive support for developing rich client applications based on the NetBeans Platform.

The NetBeans IDE is a powerful development environment and, combined with the NetBeans Platform, it allows developers to create purpose-built tools to perform specialized programming tasks.

2.3.6 Eclipse

Eclipse refers to both a software platform and an integrated development environment built on top of the Eclipse Platform [Eclipse, 2007]. It was originally developed by IBM which gave the code to the Eclipse Foundation

under an open-source license. It is written in Java and it runs on multiple platforms where the Java Virtual Machine is available.

The Eclipse Platform defines a set of frameworks and common services that collectively facilitate the creation and integration of IDE-based products and rich client applications [des Rivieres and Wiegand, 2004, Gruber et al., 2005]. The frameworks and services provide facilities [des Rivieres and Beaton, 2006] which are required by most tool builders such as:

- A platform runtime system for loading and managing plug-ins.
- A generic windowing system with portable native widget toolkits.
- Incremental project builders and compilers support.
- A language-independent debug infrastructure.
- Tools and APIs for managing workspace resources such as files, folders, and projects.
- An integrated help system.

The Eclipse Platform has been designed in a modular way. Except for the runtime system, all platform functions are implemented as plug-ins. An Eclipse plug-in is the smallest unit of the platform that implements a well-defined feature.

The Eclipse IDE is a generic integrated development environment built on top of the Eclipse Platform. It has the standard IDE user interface with multi-window editors, refactoring, syntax highlighting, and other features commonly found in advanced IDEs. It can be used to develop Eclipse-based rich client applications and software written in various programming languages such as C/C++, PHP, Fortran, Python, and Perl. The IDE can be augmented with new features due to its extensible plug-in system. Currently, there are more than one thousand Eclipse plug-ins providing rich and diverse features for software development [Yang and Jiang, 2007]. A number of notable Eclipse plug-ins provide support for parallel programming, data

modeling, embedded devices, database development, web services, and so forth.

According to Amsden [2001], Eclipse supports a number of integration levels that tool or plug-in developers can target based on the time to market, desired level of investment, and specific tool needs. In the order of increasing integration sophistication, these levels are:

- None. There is no integration of tools with Eclipse, tools are separate and independent.
- Invocation. The integration with Eclipse is through invocation of registered applications on resource types.
- Data. The tool or plug-in integration is achieved through data sharing.
- API. Tools interact with other tools through Java APIs that abstract tool behavior and make it publicly available.
- User Interface. Tools and their user interfaces are dynamically integrated at runtime including window panes, toolbars, menus, properties, preferences, etc.

Eclipse provides a complete suite of frameworks, tools, and environment for application development. Similar to NetBeans, it allows developers to create a specialized tool to suit a particular programming task.

2.4 Grid Computing

The term *grid computing* was introduced by Foster and Kesselman [1998a] in the book "The Grid: Blueprint for a New Computing Infrastructure" to denote an innovative distributed computing infrastructure for advanced science and engineering. According to Foster [2002], a grid can be defined as a system that coordinates resources that are not subject to centralized control using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service. Although grid computing is not synonymous with high

performance parallel computing, heterogeneous computing, or distributed computing, it relies heavily on the concepts and the underlying technologies that have been developed in each of these fields. Grid computing has also been described in the context of peer-to-peer system [Foster and Iamnitchi, 2003] and service-oriented architecture [Foster and Tuecke, 2005].

There are several characteristics of grids which are compiled into an extensive list by Bote-Lorenzo et al. [2003] from main grid literature. More recently, Stockinger [2007] conducted a survey to define important aspects that build a grid. The characteristics that grid researchers generally agree on are as follows: collaboration, aggregation, virtualization, service orientation, heterogeneity, decentralized control, standardization and interoperability, access transparency, scalability, reconfigurability, and security [Stockinger, 2007].

The idea of the grid has been inspired by, and was first applied to, difficulties faced by researchers in tackling fundamental problems in science and engineering [Berman and Hey, 2003]. The difficulties stem from the fact that solving these complex problems requires capacity and resources (for example, large-scale experiments, data analysis, and simulation) beyond that of individual organizations. As a result, there is an emergence of virtual organizations which are groups of individuals and/or institutions who collaborate and share collective resources for common scientific enquiries. John Taylor coined the term *e-Science* to denote this new kind of science, writing that "e-Science is about global collaboration in key areas of science and the next generation of infrastructure that will enable it" [NeSC, 2007, Berman and Hey, 2003]. e-Science has much to benefit from grid computing which addresses the problem of coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [Foster et al., 2001]. Grid computing has been successfully applied to support e-Science in various domains such as astronomy [Williams, 2003], high-energy physics [Bunn and Newman, 2003], biology [Baldrige and Bourne, 2003], medicine [Brady et al., 2003], and chemistry [Frey et al., 2003].

2.5 Service-Oriented Grids

Technological advances in areas such as virtualization software, high-speed networking systems, and the proliferation of the Internet have augmented the meaning of applications. Traditionally, an application is composed of source code files that need to be compiled into an executable program according to the runtime platform. The program can then be executed or submitted to a job scheduler. This type of application is statically bound to the underlying system. Executing an application involves the creation of an operating system process.

In the service-oriented model, an application is represented as a composition of services which can be distributed over a network or the Internet. A service is defined as a network addressable entity that provides some functions. These services perform the application logic and communicate by exchanging messages from one service to another. Thus, these services can largely be implemented in different programming languages and executed on various machines as long as they share the same communication protocol.

This loosely coupled architectural style is attractive to grid computing which is characterized by heterogeneity, decentralized control, scalability, and interoperability [Foster et al., 2003b, Foster and Tuecke, 2005, Foster et al., 2006]. This section describes service-oriented grids in detail, starting from web services (Section 2.5.1), followed by Web Services Resource Framework (Section 2.5.2) and Open Grid Services Architecture (Section 2.5.3).

2.5.1 Web Services

Web services are of special interest in grid computing. They are platform and implementation independent software components and their function of interconnecting information systems is similar to the intended function of grid computing [Atkinson, 2003]. The World Wide Web Consortium (W3C) defines a web service as a software system which is designed to support interoperable machine-to-machine interaction over a network [W3C, 2004]. Web services utilize common protocols and standards that allow them to communicate with one another regardless of the deployment platforms or

the implementation languages. The primary protocol for web services is the HyperText Transfer Protocol (HTTP) which is a stateless protocol used for data communication on intranets and the Internet.

A number of specifications have been defined for web services by various standards organizations. A list of these specifications can be found in W3C [2008], OASIS [2008a], IBM [2008]. Specifications which are deemed to form the basis of web services are:

- Extensible Markup Language (XML). It is a simple general purpose text format that facilitates the sharing of structured data [W3C, 2006]. It is designed to be platform independent, human legible, and easy to process by computer programs.
- SOAP. It is an XML based protocol for exchanging information between computers in a distributed environment [W3C, 2007a]. Fundamentally, it is a stateless, one-way message exchange specification. SOAP was originally an acronym for Simple Object Access Protocol. SOAP is a successor to XML-RPC which is a simple remote procedure call protocol that uses HTTP as the transport mechanism to deliver XML encoded messages [Winer, 1999].
- Web Services Description Language (WSDL). It is an XML based language for describing web services [W3C, 2007b]. A WSDL description for a service specifies message types that the service can send and receive; the service location; and the mechanism on how to access the service.

The importance of web services is further strengthened with the adoption of the Web Services Resource Framework (WSRF), which is described in the following section, as the supporting foundation for grid middleware architecture [Foster et al., 2003a, WSRF, 2008].

2.5.2 Web Services Resource Framework

Traditional web services are stateless, which prevents them from retaining data between invocations. Various workarounds exist, such as browser cook-

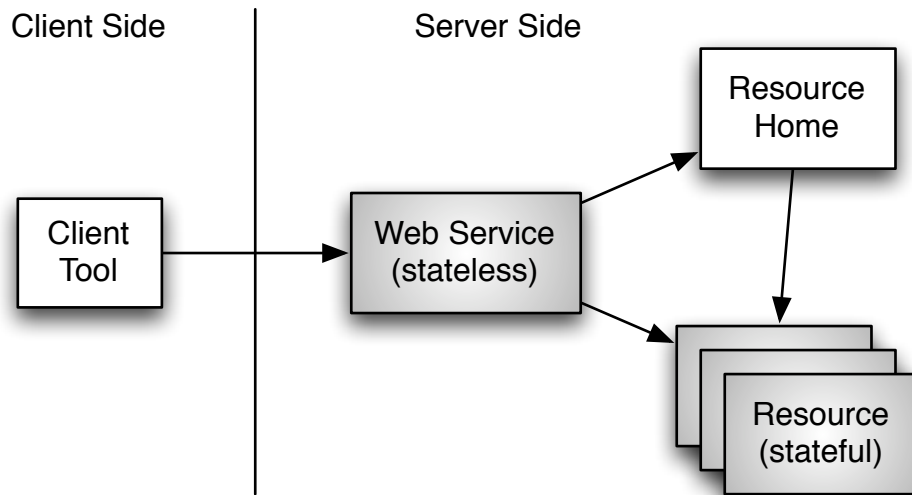


Figure 2.1: Web service and WS resources relationship

ies and session identifications, to enable stateful web services. Nevertheless, these workarounds are non-standard and may not allow communication between services.

The Web Services Resource Framework (WSRF) is a set of specifications that provides a standardized way to create and access stateful web services [WSRF, 2008, OASIS, 2008b]. WSRF is defined by a standards body, Organization for the Advancement of Structured Information Standards (OASIS). The specifications allow web services to retain their states while communicating with one another or with web service clients.

A web service retains its state information for a client in a separate entity called a resource. The web service utilizes another entity called a resource home to create, find, and manage web service resources. Web service operations are invoked on the resources. Each resource has a unique identifier, thus, it allows the client to instruct the web service to use a particular resource. Figure 2.1 illustrates the relationship between a client, a web service, a resource home, and a collection of web service resources.

Web Services Resource Framework is comprised of a number of specifications:

- WS-Resource. It defines a relationship between a web service and the

WSRF concept of a resource; methods to access resources through web services; and means by which WS-Resources are referenced [OASIS, 2006a].

- **WS-ResourceProperties.** It defines properties of WS-Resources which can be accessed and updated through web service interfaces [OASIS, 2006b]. These properties represent the state of particular WS-Resources.
- **WS-ResourceLifetime.** It defines interfaces to manage, inspect, and monitor the lifetime of WS-Resources [OASIS, 2006c]. Web service resources can be destroyed immediately or using a time-based mechanism.
- **WS-ServiceGroup.** It defines a mechanism for managing service groups [OASIS, 2006d]. A service group can be used to form a collection of web services or WS-Resources.
- **WS-BaseFaults.** It defines XML based information which may appear in fault messages or exceptions returned by web services or produced by WS-Resources [OASIS, 2006e].

WSRF provides the specification for stateful web services which form the basis for the Open Grid Services Architecture (OGSA). The next section gives a description of OGSA.

2.5.3 Open Grid Services Architecture

The Open Grid Services Architecture (OGSA) is a service-oriented architecture (SOA) that specifies a standard set of core capabilities and behaviours that define a grid system [Foster et al., 2003b, 2006]. These capabilities address key concerns such as job management, resource management, security, and so forth. OGSA is based on web service technologies, especially WSRF. According to Sotomayor and Childers [2005], the relationship can be expressed by describing WSRF as the *infrastructure* on which OGSA is built. OGSA has also been described by Gannon et al. [2002], as an extension and a

refinement of web services architecture, specifically designed to support grid systems.

Within OGSA, everything is represented as a grid service which is defined as a web service that conforms to specific conventions and provides a set of well-defined interfaces to comply with additional grid requirements. This service-oriented model encapsulates diverse implementations into consistent interfaces and facilitates the construction of higher-level services that can be accessed uniformly across various abstraction layers [Foster et al., 2003a].

Notable examples of grid middleware that implement high-level services according to the abstract requirements described in OGSA are: Globus Toolkit 4 [Foster, 2005a,b, Sotomayor and Childers, 2005] and UNICORE 6 [Menday and Wieder, 2003, Snelling, 2003, UNICORE, 2008]. In spite of the development of OGSA, a number of non service-oriented grid middleware are still in wide use and under continuous development and maintenance, for example, Globus Toolkit 2, UNICORE 5, and gLite [gLite, 2008]. This is due to the large number of existing scientific programs written in legacy languages such as C/C++ and Fortran. It may take significant investment and time to convert these legacy applications and libraries to grid services. We believe that the development of traditional grid applications, in the form of program executables as opposed to grid services, will continue to grow in spite of the push towards service-oriented architecture.

2.6 Grid Middleware

Grids that comprise several organizations are inherently large-scale, distributed, and heterogeneous. These attributes and other concerns such as data access and authentication must be confronted by grid application developers. To relieve programmers from some of these concerns, the grid community has developed specific middleware that provides a uniform set of services for security, information, and management. Middleware can be defined as the software layer between the operating system and applications, providing common high-level functions required by applications to operate correctly. In a grid, the middleware is used to hide the heterogeneous nature

and provide users and applications with a homogenous and seamless environment by implementing a set of standardized interfaces to a variety of services [Roure et al., 2003].

Grid middleware services can be categorized into two types: low-level and high-level [von Laszewski and Amin, 2004, Asadzadeh et al., 2005, Abramson et al., 2006b]. von Laszewski and Amin [2004], distinguished between elementary and advanced grid middleware services, while Asadzadeh et al. [2005] used different terminologies: core and user-level. This distinction is due to the fact that typical middleware services, whilst powerful, are too low-level for grid developers and e-Scientists to use, and thus it is necessary to provide higher level abstractions and services.

Similarly, Abramson [2006] proposed a hierarchy of grid middleware which consists of lower and upper layers. The lower middleware provides essential functionalities such as user authentication, authorization, data storage management, and information services. The upper middleware and software tools which are built on top of the lower layer provide high-level functionalities and further enhance the ease of use of grid middleware for scientific and engineering applications.

Representative examples of grid middleware are described in the following sections: Section 2.6.1 introduces Globus Toolkit which is the de facto standard for open source grid middleware, Section 2.6.2 describes UNICORE which is a vertically integrated middleware in widespread use in Europe, and Section 2.6.3 introduces gLite which is the grid middleware stack developed by the EGEE project.

2.6.1 Globus Toolkit

Globus Toolkit [Foster and Kesselman, 1997, 1998b] is a popular grid middleware stack developed by the Globus Alliance. It pioneered the creation of interoperable grid systems and it has emerged as the de facto standard for grid computing [Foster and Kesselman, 2003]. The toolkit is constructed in a layered fashion where high-level services are built upon low-level core services.

Globus Toolkit version 2 (GT2) which was released in 2002 is an important and early implementation of the toolkit. It is the foundation for many major grid projects such as the National Grid Service in UK [NGS, 2007] and NorduGrid [NorduGrid, 2007]. GT2 is composed of three primary components: Resource Management, Information Services, and Data Management, that utilize a common security foundation, Security Infrastructure [GT2, 2007]:

- Resource Management. GRAM which stands for Globus Resource Allocation Manager implements a resource management architecture for GT2 [Czajkowski et al., 1998]. It provides a single standard interface to various local resource management tools for requesting and using remote resources for the execution of jobs.
- Information Services. The Globus Metacomputing Directory Service (MDS) provides scalable and efficient access to dynamic information about resource structure and state [Fitzgerald et al., 1997, Czajkowski et al., 2001].
- Data Management. GridFTP is responsible for the data management in GT2 [Allcock et al., 2001, 2002]. It is a universal grid data transfer and access protocol based on the standard FTP protocol. GridFTP provides a secure and efficient data transport mechanism with new features required for data grid applications, such as striping and partial file access.
- Security Infrastructure. GT2 uses the Grid Security Infrastructure (GSI) as the underlying security foundation for GRAM, MDS, and GridFTP [Foster et al., 1998, Butler et al., 2000]. It is based on public key encryption, X.509 certificates, and the Secure Socket Layer (SSL) protocol. It provides services for single sign-on, authentication, authorization, delegation, and encrypted communication.

Globus Toolkit version 4 (GT4), which was released in April 2005, is an evolution of previous Globus Toolkit versions. It implements web services

mechanisms for building grid systems and meets the requirements of OGSA [Foster, 2005a,b, Sotomayor and Childers, 2005]. GT4 components fall into five domain areas: Security, Data Management, Execution Management, Information Services, and Common Runtime:

- **Security.** GT4 implements standard security protocols that address secure communication, authentication, authorization, and delegation. It implements WS-Security mechanism with X.509 certificate credentials.
- **Data Management.** GT4 provides the Reliable File Transfer (RFT) service which is a WSRF-enabled service to manage large amounts of data. It also provides the Replica Location Services (RLS) for managing data replication.
- **Execution Management.** WS Globus Resource Allocation Manager (WS GRAM) in GT4 provides services for the deployment, scheduling, and monitoring of executable programs.
- **Information Services.** GT4 offers two aggregator services that collect statistics and data from registered information providers. The aggregators implement a registry (Index Service) and event-driven data filters (Trigger Service).
- **Common Runtime.** A wide range of enabling software such as libraries and tools are included in GT4. They allow the development and hosting of new services written in C, Java, and Python.

2.6.2 UNICORE

UNICORE (Uniform Interface to Computing Resources) is a vertically integrated grid middleware which provides seamless, secure, and intuitive access to distributed resources and data [Erwin and Snelling, 2001, UNICORE, 2008]. It was started in 1997 as a German national research project to develop a software infrastructure that provides uniform access to computing resources in various cities with minimal intrusion into local site policies. UNICORE

follows a client-server architecture that consists of three tiers, namely User, Server, and Target System [Romberg, 1999, 2002]:

- User tier. It consists of GUI client tools which support the creation, manipulation, and control of UNICORE jobs. It utilizes SSL-based communication protocol and X.509 certificates as the authentication mechanism.
- Server tier. The middle tier consists of a gateway and a Network Job Supervisor (NJS). The gateway acts as a secure entry point into a UNICORE site and authenticates users based on their certificates. It also provides UNICORE clients with information about resources. The gateway communicates with the NJS, which translates abstract user requests prepared using the client tools, into concrete jobs to be executed by target systems. It also performs additional tasks such as file staging and data collection.
- Target System tier. It consists of a small daemon called the Target System Interface (TSI) which communicates with the underlying local resource management system to spawn and execute jobs.

In recent years, UNICORE has evolved towards a service-oriented architecture compliant to the OGSA and WSRF standards [Menday and Wieder, 2003, Snelling, 2003, Streit et al., 2005, Menday, 2006]. It still retains its seamless, secure, and intuitive characteristics, however, the coupling between the components of the system has been loosened to allow greater interoperability with other WSRF-compliant software components. For example, this development allows jobs launched via UNICORE client tools to be executed on Globus resources. UNICORE has been used as the base grid middleware system in many European and international research projects in areas such as astronomy, climate modeling, high-energy physics, and molecular science [Streit et al., 2005, Rambadt et al., 2007].

2.6.3 gLite

gLite is the grid middleware stack developed as part of the EGEE (Enabling Grids for E-scienceE) project, a major grid initiative funded by the European Union. The project integrates various national, regional, and thematic grid deployments into a common pool of resources for collaborative compute-intensive science. The aim of the EGEE project is to provide a seamless and high-quality service to multiple scientific communities, through the development of a production-quality grid infrastructure [Jones, 2004, Berlich et al., 2005, Gagliardi et al., 2005, gLite, 2008].

The middleware deployed on EGEE sites is gLite. It is built by leveraging the experience and components from existing grid middleware, in particular Globus, Condor [Condor, 2007], and VDT [VDT, 2007]. gLite provides high-level services for accessing and moving data, scheduling and running computational jobs, and obtaining information on the grid infrastructure [Laure et al., 2004, 2006]. gLite follows a service-oriented architecture that facilitates interoperability among grid services. gLite services can be categorized into 5 service groups:

- Access Services. This service group serves as the back-end for gLite web portals and it also provides a common interface for gLite services.
- Security Services. This service group encompasses Authentication, Authorization, and Auditing Services. They enable the identification of entities, grant or deny access to services and resources, and provide logging information.
- Job Management Services. This group provides services related to job management and execution. They are responsible for virtualizing computing resources, scheduling jobs, and keeping track of job-related information such as user inputs and result outputs.
- Information and Monitoring Services. This service group provides a mechanism to publish and inquire information concerning grid resources. The information can also be used for monitoring purposes.

- **Data Services.** The three main services in this group are Storage Element, File and Replica Catalog, and Data Management Services. They are responsible for providing virtualization of storage resources, registering file data and metadata, and managing data transfers.

gLite is currently deployed on a large number of EGEE sites involved in many scientific projects [Gagliardi et al., 2005]. It is also used as the standard grid middleware stack in the LHC Computing Grid (LCG) project [LCG, 2007].

2.7 Grid Application-Level Tools

Grid middleware provides fundamental services for security, information, and management. However, these services are too low-level for e-Scientists to use directly [Bal et al., 2003]. They may not have the time or expertise, for example, to learn the intricacies of grid information services to obtain and use resource information. In addition to this difficulty, there is also a consensus that current programming tools are inadequate for the grid environment [Lee et al., 2001, Bal et al., 2003, Balle and Hood, 2004].

Grid computing requires sophisticated application-level tools that can hide the low-level details of the middleware. According to Bal et al. [2003], a grid application-level tool can be defined as software that is built on top of the base grid infrastructure to provide new functionalities and high-level abstractions that can be employed by users to easily write and/or run grid applications. Specifically, these tools must meet a number of criteria: provide higher-level abstractions, isolate users from the dynamics of the grid infrastructure, be applicable to broad classes of applications, and be easy to use [Bal et al., 2003].

There are a number of research papers that survey and offer classifications of these tools [Lee et al., 2001, Laforenza, 2002, Fox et al., 2003, Lee and Talia, 2003, Bal et al., 2003, Kielmann et al., 2005, Parashar and Browne, 2005, Soh et al., 2006]. However, they may not use the term *grid application-level tools* as employed by Bal et al. [2003] and this thesis. For example, Lee et al.

[2001], Laforenza [2002], Lee and Talia [2003], Soh et al. [2006] use the term *grid programming models or methodologies*.

Grid application-level tools can be classified into three major categories. The following three sections present detailed descriptions and surveys of the tool categories. Specifically, they are: grid programming libraries (Section 2.8), grid application execution environments (Section 2.9), and grid development environments (Section 2.10).

2.8 Grid Programming Libraries

In order to build grid applications for a federation of distributed and heterogeneous resources, there is a strong need for high-level programming middleware or libraries that can interface directly to application codes and provide higher-level functions [Kielmann et al., 2005]. This thesis uses the term *grid programming libraries* rather than *programming middleware* to avoid the confusion with *grid computing middleware*. Grid programming libraries should hide the heterogeneity and complexity of the underlying infrastructure and provide seamless access for the application codes to computational and data resources. Typically, the libraries are mapped directly to low-level grid middleware APIs or they may provide runtime environments for the execution of library interfaces. Based on the type of programming models, grid programming libraries can be broadly categorized into three: message passing libraries (Section 2.8.1); object-based and component-based libraries (Section 2.8.2); and remote procedure call libraries (Section 2.8.3).

2.8.1 Message Passing Libraries

The message passing paradigm is one of the most general and widely used programming models for parallel and distributed computing [Bal et al., 2003]. It provides send and receive primitives that applications can use to exchange data and synchronize. It supports point-to-point communication between two processes and collective operations such as broadcast and reduction that involve multiple processes. There are several implementations of message

passing libraries for grid computing.

PACX-MPI [Keller et al., 2003] use vendor MPI (Message Passing Interface) [MPI Forum, 1997, Gropp et al., 1998] libraries with a hierarchy of communication layers for message exchange between grid nodes. MagPie [Kielmann et al., 1999] implements collective communication primitives which are optimized for grid environments. MPICH-G2 [Karonis et al., 2003] is a grid-enabled MPI library that utilizes various services provided by Globus Toolkit 2. A paper by Muller et al. [2003] describes these three libraries in detail and provides some performance measurements.

2.8.2 Object-Based and Component-Based Libraries

Object-based and component-based libraries allow users to decompose a complex application into smaller code units that are conceptually more manageable. These units or modules are accessed via well-defined interfaces and they implement specialized functions and interact in specific ways. There are a large number of projects that offer object-based and component-based programming models for grid computing.

ICENI [Mayer et al., 2004] provides an augmented component programming model with metadata to support grid applications. The metadata system is used to describe the meaning, behaviour, and implementation of a component. XCAT [Krishnan and Gannon, 2004, Gannon et al., 2004] implements a software framework that models a component as a set of grid services. ASSIST [Aldinucci et al., 2005, 2006] is a programming framework that structures a grid application as a graph, whose nodes correspond to parallel or sequential modules and the graph arcs correspond to data flow streams between modules. Cactus [Goodale et al., 2002] is a modular framework for building a variety of computing applications. A cactus application consists of a core component which is called the flesh and multiple modules which are called thorns. The flesh acts as an execution entry point and manages the thorns which implement the application functionalities. The Commodity Grid Toolkits (CoG Kits) [von Laszewski et al., 2003] aim to simplify grid application programming by providing interfaces between grid

services and particular commodity frameworks. The Java CoG Kit provides client-side access to various Globus services in an object-oriented manner.

ProActive [Caromel et al., 2006] is a Java library and middleware for programming and running applications on grids and peer-to-peer systems. It is based on the concept of active objects. An active object is the basic unit of activity that has its own thread and execution queue. Ibis [van Nieuwpoort et al., 2005] is a Java-based programming environment that provides a high-level communication API that hides the complexity of grid and fits into Java's object model. Grid Application Toolkit (GAT) [Allen et al., 2005] aims to simplify grid programming by providing a simple object-oriented API with well-known paradigms. A research paper that compares ProActive, Ibis, and GAT in detail is given by Kielmann et al. [2005].

2.8.3 Remote Procedure Call Libraries

The Remote Procedure Call (RPC) programming model supports communication and cooperation between applications by extending the traditional notion of procedure calls to operate across the network [Parashar and Browne, 2005]. The RPC model provides a well-understood and simple mechanism for performing remote computations and managing the flow of control and data [Lee and Talia, 2003]. Several projects have been initiated to offer RPC libraries that simplify access to remote computational services on grid resources.

The GridRPC Working Group represents an ongoing effort to standardize and implement a simple RPC mechanism for grid computing [Seymour et al., 2002, GridRPC-WG, 2008]. Ninf-G [Nakada et al., 2003, Tanaka et al., 2003] and GridSolve/NetSolve [YarKhan et al., 2006a,b] are two RPC systems that provide the implementation of GridRPC API. OmniRPC [Sato et al., 2003] is a thread-safe RPC system that has been designed to support seamless parallel programming in clusters and grids.

Simple API for Grid Applications (SAGA) [Goodale et al., 2006, Jha et al., 2007, SAGA-RG, 2008] is a recent standardization effort that aims to define and implement a simple, stable, and uniform API that integrates the

most common grid programming abstractions. SAGA does not try to cover all use cases and it has been designed with the 80:20-rule, where 80% of the functionality is satisfied by 20% of the effort.

2.9 Grid Application Execution Environments

In addition to programming libraries, there is a clear need for tools that allow users to deploy and run their applications on grids [Bal et al., 2003]. These tools, which are termed grid application execution environments, can be defined as software systems that support and simplify the execution of applications on distributed grid resources. For example, they may provide environments for running an application with different data sets or conducting an experiment using a number of different smaller programs. The systems handle resource logistics such as discovery, selection, and access; and application logistics such as deployment, execution, and monitoring [Bal et al., 2003]. The execution environments can be classified into three:

- Grid portals, which enable users to easily access grid resources for program execution and aggregate experiment results and information into a single web interface (Section 2.9.1).
- Workflow management systems, which provide an infrastructure for creating and executing an application composed from independent software components (Section 2.9.2).
- Parameter sweep systems, which allow users to execute the same application multiple times with variations of input parameters (Section 2.9.3).

2.9.1 Grid Portals

A grid portal can be described as a web-based gateway that provides transparent and easy access to back-end grid resources [Li and Baker, 2006]. Grid portals have become increasingly popular as interactive platforms that provide users with uniform and simple interfaces for accessing grid resources

[Thomas and Boisseau, 2003, Novotny et al., 2004]. An appealing factor of grid portals is their user-friendly attribute due to the fact that they are accessible via web browsers without the need to install specialized software. This section discusses two grid portal projects: GridSphere [Novotny et al., 2004, Russell et al., 2005] and the NPACI Grid Portal Toolkit (GridPort) [Thomas et al., 2001, Thomas and Boisseau, 2003]. Research papers that survey and compare major grid portal tools are provided by Li and Baker [2006], Yang et al. [2006], Zhang et al. [2007].

GridSphere is a Java-based framework for developing grid portals [Novotny et al., 2004, Russell et al., 2005, GridSphere, 2008]. It is part of GridLab which is a European research project for the development of application tools and middleware for grid environments [GridLab, 2008]. GridSphere is designed to be modular and flexible to support different needs and requirements of portal users and service providers. The modularity of GridSphere is achieved through the use of components called portlets which are standardized in JSR 168 specification [Java Community Process, 2008]. Portlets are user interface components implemented as Java classes that provide certain functions in a web portal. GridSphere offers a service registry and essential portlet services that can be leveraged to build customised and sophisticated grid portals for e-Scientists.

GridPort is the grid portal software toolkit used by the National Partnership for Advanced Computational Infrastructure (NPACI) project [Thomas et al., 2001, Thomas and Boisseau, 2003, GridPort, 2008]. It is an evolution of NPACI HotPage which was intended to provide users with a web-based portal to access NPACI resources. GridPort has been generalized to provide a flexible portal infrastructure for scientific community. It has a multi-layer architecture that provides capabilities such as user accounts, authentication, job execution, data management, and information services. In recent years, the GridPort project has become part of the Open Grid Computing Environments (OGCE) project [Alameda et al., 2007, OGCE, 2008]. The OGCE project is a collaborative effort to develop a standard set of software components such as JSR 168 portlets, web services, libraries, and tools for building grid portals.

2.9.2 Workflow Management Systems

A workflow is concerned with the automation of procedures where participants cooperate to achieve an overall goal by processing files and data in a pipeline manner according to a defined set of rules [Hollingsworth, 1995]. A grid workflow management system is responsible for defining, managing, and executing workflows on computing resources. Workflow management has been the subject of many research projects and it is increasingly popular in the grid computing area. This is due to the fact that workflow's component architecture encourages software reuse and it is a natural way to utilize services that promote collaboration between organizations [Spooner et al., 2005]. Representative examples of workflow management systems are Kepler [Ludascher et al., 2006], Triana [Churches et al., 2006], and Taverna [Oinn et al., 2006]. A comprehensive survey of grid workflow systems can be found in Yu and Buyya [2006] whilst The Grid Workflow Forum maintains a list of grid workflow projects [Grid Workflow Forum, 2008].

Kepler is an open-source workflow system that can be used to design scientific workflows and execute them on distributed grid resources [Ludascher et al., 2006]. It is based on Ptolemy II which is a Java-based component assembly framework for concurrent modeling and design [Ptolemy II, 2008]. In Kepler, a scientific workflow is a composition of independent components called actors. A Kepler actor has ports for input and output; and parameters for configuring the behaviour. A channel connects two or more actors through their ports. Kepler allows the execution semantics of a workflow to be specified by an object called a director which defines how actors are executed and how they communicate with one another. A Kepler director relieves the actors from the details of component interaction and improves the reusability of actors. Kepler provides actors for accessing web services and grid actors that facilitate workflows which utilize grid resources. Kepler has been successfully used in many scientific domains such as biology, geology, and chemistry [Ludascher et al., 2006].

Triana is a workflow system that provides an environment for the composition of scientific applications from independent programming components

[Churches et al., 2006]. A Triana component is a Java class that represents the smallest granularity of work that can be performed. It consists of an identifying name, input and output ports, optional name/value parameters, and a process method. Triana has a highly decoupled modularized architecture and it can dynamically discover and choreograph resources such as web services, peer-to-peer (P2P) nodes, and grid resources. Triana can distribute a group of work components across multiple machines either in parallel or in a pipeline. It uses Grid Application Prototype (GAP), a high-level middleware independent interface, for interacting with the underlying infrastructure. GAP provides an abstraction that hides low-level details such as job submission and resource monitoring from the workflows.

Taverna enables e-Scientists to create and run workflows that link together third-party applications using a language and tools familiar to the scientific users [Oinn et al., 2006]. It is a user-centric application that can be used to access remote grid resources. Taverna places an emphasis on handling a variety of autonomous service providers and it can utilize services without requiring the providers to change the interface. Taverna provides an extensible framework that facilitates integration with external services and a data model for describing resources at different levels of abstraction, from the user perspective to the service perspective. Taverna workflows are written in Scuff (Simplified Conceptual Workflow Language) and enacted using the Freefluo workflow enactment engine. Scuff is a data flow language for describing Taverna workflow components and the service interaction. The main user interface is provided by the Taverna Workbench which is a data-model centric GUI for the construction and execution of Scuff workflows.

2.9.3 Parameter Sweep Systems

A parameter sweep application can be defined as a set of related computational tasks where the tasks are mostly independent from one another [Casanova and Berman, 2003]. The tasks have few data dependencies or task-synchronization requirements. The loose coupling between tasks make parameter sweep applications suitable for peer-to-peer (P2P) or grid comput-

ing. As noted by Casanova and Berman [2003], in spite of the simplicity of their execution model, parameter sweep applications have been widely used in many science and engineering fields. This section summarises two popular parameter sweep systems: Nimrod/G [Abramson et al., 2000a] and APST [Casanova and Berman, 2003].

Nimrod/G is a grid-enabled system for the execution of parametric studies across distributed grid resources [Abramson et al., 2000a]. It manages various aspects of an automated parametric experiment such as distributing input files to compute nodes, executing the experiment, and collecting the results. The experiment is described using a plan file written in a simple declarative modelling language. The file specifies parameter values of the experiment and task descriptions that need to be performed. One job is generated for each unique combination of the parameter values, by taking the cross product of all values. A tool called the Nimrod/G generator processes the plan file and produces a run file, which contains a description for each job, as an input to another tool, the dispatcher. The Nimrod/G dispatcher is responsible for preparing and dispatching parametric jobs across the nodes. The tasks of resource selection and job allocation are performed by the Nimrod/G scheduler. It provides several scheduling heuristics based on monetary cost, time, and resource availability. Nimrod/G has been employed in a range of application areas such as bioinformatics, network simulation, and climate research [Abramson et al., 2000a, 2006a, Faux et al., 2007]. Nimrod/G can also be augmented with Nimrod/O that offers a number of optimization algorithms to search the parameter space more efficiently [Abramson et al., 2000b, Peachey et al., 2003, Abramson et al., 2006c].

The AppLeS Parameter Sweep Template (APST) project aims to provide an easy and effective deployment of parameter sweep applications for grid users [Casanova and Berman, 2003]. It handles deployment logistics such as resource discovery, data transfer, application launching, and monitoring. APST consists of two distinct components: a client and a daemon. The client is the APST user interface that can be invoked to submit experiments and check on the execution progress. The daemon is the APST server component that consists of a scheduler and three APST managers: data, compute, and

metadata managers. The scheduler is responsible for allocating grid resources for computational tasks and data management. The allocation is based on the information provided by the managers. The data and compute managers launch and monitor data transfers and computations. The metadata manager obtains information such as resource availability and CPU information for the scheduler. APST leverages various services provided by Globus such as GRAM, GridFTP, GSI, and MDS.

2.10 Grid Integrated Development Environments

The third category of grid application-level tools is grid integrated development environments. They consist of development tools that can be used by programmers and users to write and test grid software. They also allow grid developers to find application bugs and investigate the execution performance of the software. It is not impossible to develop grid software using traditional IDEs, however, the grid application development process is sufficiently different to warrant specialized grid IDEs.

There are several integrated development environments which are specifically designed for writing grid applications. Indicative examples that represent the current state of research are: P-GRADE (Section 2.10.1), GT4IDE (Section 2.10.2), GriDE (Section 2.10.3), g-Eclipse (Section 2.10.4), GDT (Section 2.10.5), and Introduce (Section 2.10.6).

2.10.1 P-GRADE

P-GRADE (Parallel Grid Runtime and Application Development Environment) [Kacsuk et al., 2003] from MTA SZTAKI in Hungary provides a high-level graphical environment to develop parallel applications for parallel systems and grid. It consists of an editor (Gred) for a graphical language (Grapnel), a debugger (Diwide), a profiler (Prove), and application monitoring tools (GRM and Mercury). The aim of P-GRADE is to hide the low-level details of parallel APIs from programmers by generating the necessary

application code according to the actual execution platforms. P-GRADE supports several APIs: PVM (Parallel Virtual Machine) [Geist et al., 1994], MPI (Message Passing Interface) [MPI Forum, 1997, Gropp et al., 1998], and GAT (Grid Application Toolkit) [Allen et al., 2005].

In P-GRADE, rather than directly write an application in C/C++ or Fortran, developers write the program graphically in Grapnel using Gred. Grapnel is a hybrid programming language that uses both textual and graphical representations to describe the application. P-GRADE converts the Grapnel code to C/C++ or Fortran code, links it with the necessary library, and compiles it into an executable. The resulting application can be submitted to resources as Condor, Condor-G, or Globus 2 jobs. Debugging PVM and MPI applications can be performed by using Diwide. It utilizes a macrostep debugging approach [Kacsuk, 2000] to facilitate parallel debugging. Performance monitoring and profiling are performed by using GRM, Mercury, and Prove tools that support both PVM and MPI programs. The tools also provide performance visualization based on the output trace files.

2.10.2 GT4IDE

GT4IDE [GT4IDE, 2008] is an integrated development environment specifically suited to the needs of Globus Toolkit 4 (GT4) programmers. It is built on top of Eclipse IDE. The main target of GT4IDE is the development of grid applications as services, specifically GT4 services. It presents an environment that seamlessly integrates all the development steps from coding to deployment. However, it does not support debugging and profiling of grid services. GT4IDE provides the necessary build tools and automatic generation of WSDL (Web Services Description Language) files for GT4 grid services. It can generate a compressed package file which consists of deployment description files, stub classes, and the Java archive file. The package can then be manually deployed on any GT4 containers. Although GT4IDE simplifies service deployment for GT4 programmers, the project is not under active development anymore.

2.10.3 GriDE

GriDE [See et al., 2003] from Asia Pacific Science and Technology Center (APSTC) is a grid-enabled IDE utilizing NetBeans as the base IDE and Globus Toolkit 2 as the grid middleware layer. Its goal is to provide users and developers an easy to use integrated environment to develop, debug, run, and monitor grid applications. GriDE provides a traditional programming environment through NetBeans and a set of grid-specific tools. Particularly, the prototype implementation offers a grid resource browser, a grid job submission tool, a job monitoring tool, and a simple workflow editor.

Grid users can browse and search for available grid resources using the resource browser. It retrieves resource information from various directory services such as Globus Metacomputing Directory Service (MDS) and Lightweight Directory Access Protocol (LDAP) servers. Within GriDE, users can submit GRAM jobs to remote Globus gatekeepers directly. GriDE generates the necessary Globus Resource Specification Language (RSL) scripts for the GRAM jobs. The jobs can be executed interactively or in batch mode. After the submission, the jobs can be monitored using GriDE's job monitoring tool. Execution results and error streams can also be retrieved and displayed by the tool. GriDE's grid workflow editor can be used to construct simple workflow like batch job and single job executions. The editor provides various workflow templates and generates submission scripts based on user-given parameters. Functionalities such as grid debugging and complex grid workflows are still not implemented in the current prototype.

2.10.4 g-Eclipse

The g-Eclipse [Stumpert, 2007, g-Eclipse, 2009] project aims to build an integrated workbench tool for grid computing on top of the Eclipse IDE. It is developed by a collaboration of several European universities under the g-Eclipse consortium. g-Eclipse offers a user-friendly development environment for grid programmers by hiding the complexity of grid infrastructure. Its architecture has been designed to be extensible and reusable with well-defined interfaces.

g-Eclipse distinguishes three distinct user roles, namely application users, grid operators, and developers. It provides different Eclipse perspectives for each role. An Eclipse perspective is a collection of related windows and views for a specific task.

- The user perspective allows a grid application user to submit and monitor computational jobs. It also provides tools for data management and visualization of scientific results.
- The operator perspective offers tools for configuring, managing, and maintaining various entities such as grid resources, storage elements, and virtual organizations. Additionally, it includes resource monitoring and benchmarking tools.
- The developer perspective allows a grid programmer to code, execute, and debug grid applications. It offers a command-line console for developers to access remote grid resources.

g-Eclipse has been designed with a middleware-independent model and it supports several grid middleware such as gLite [gLite, 2008] and GRIA [GRIA, 2009]. In addition, g-Eclipse can also be used to manage and access cloud computing resources such as Amazon Web Services [AWS, 2009].

2.10.5 GDT

Grid Development Tools (GDT) [Friese et al., 2006a,b] is a set of Eclipse IDE plug-ins for service-oriented grid application development. It is part of the Marburg Ad-hoc Grid Environment (MAGE) which is a WSRF-compliant grid middleware solution developed in the University of Marburg in Germany. GDT is targeted at non-grid programmers and it allows them to rapidly develop grid applications without knowing the intricacies of the middleware. GDT follows a model-driven approach [Kleppe et al., 2003, Frankel, 2003, Brown, 2004] by dividing the development process into three separate model layers and automatically transforming models from one layer into the other. The three model layers are described as follows:

- The Platform Independent Model (PIM) layer holds a formal high-level representation of the entire software. It does not hold any references to specific operating systems, grid middleware, or programming languages.
- The Platform Specific Model (PSM) layer contains a software representation that targets a specific platform or middleware such as Java Virtual Machine (JVM) or Globus Toolkit.
- The Code layer provides a complete specification of the software including source code and supporting files which can be compiled into an executable program.

Similar to a typical model-driven software development, grid programmers specify applications in a platform-independent manner using Unified Modeling Language (UML). GDT transforms the UML models into specific grid models using Model Transformers (MT). The model to source code conversion is performed by Code Emitters (CE) while Code Interpreters (CI) provide the reverse transformation from source code to model.

GDT is geared towards service-oriented grids similar to GT4IDE and it supports the development of GT4 and MAGE services. GDT utilizes the standard remote Java debugging in Eclipse to support interactive service debugging. The debug component allows grid users to connect and debug a target JVM that executes a grid service.

2.10.6 Introduce

Introduce [Hastings et al., 2007] is an open-source extensible toolkit that provides tools and an environment for the development and deployment of strongly typed, secure grid services. These services consume and produce data with explicit and well-defined types. It enables syntactic interoperability among grid services. Introduce leverages Globus Toolkit 4 as the underlying grid middleware to facilitate the development of WSRF-compliant services. There are two main goals of Introduce.

The first is to simplify and reduce the effort required to implement grid services, especially for e-Scientists. Introduce hides and encapsulates low-level details of the underlying middleware from e-Scientists by providing high-level templating tools. It also provides the Introduce Graphical Development Environment (GDE) to create, modify, and deploy grid services. The runtime support for GDE is provided by the Introduce engine which consists of Service Creator, Service Synchronizer, and Service Deployer.

- The Service Creator is composed of a number of templates using the Java Emitter Templates (JET) engine [Popma, 2004a,b]. It produces the required configuration files, deployment description files, and source code directory structures for a grid service.
- The Service Synchronizer maintains a consistent relationship between the source code and the service description. It manages and validates the Web Services Description Language (WSDL) files of a service.
- The Service Deployer generates a grid archive (gar) file which contains description files, libraries, and compiled code. The package file can then be registered and deployed in a GT4 container.

The second goal of Introduce is to enhance interoperability in the grid environment by providing support for the development of strongly typed services. The services publish the information about input and output parameters in a grid-enabled data type repository server. By using well-defined and published data types, it increases the level of compatibility and interoperability among grid services. Introduce utilizes XML schemas as the message exchange format for publishing and discovering service data types. An XML schema describes the structure of data elements that are exchanged between two services.

2.11 Background and Literature Review Summary

This chapter has presented background information and a literature review on the process of grid application development. This includes a discussion on a software life cycle (Section 2.1); a survey of various application development tools (Section 2.2) and integrated development environments (Section 2.3); grid computing (Section 2.4); service-oriented grids (Section 2.5); and grid middleware (Section 2.6). In addition, this chapter has discussed and surveyed various grid application-level tools (Section 2.7) that can be classified into three major categories: grid programming libraries (Section 2.8); grid application execution environments (Section 2.9); and grid integrated development environments (Section 2.10).

Software almost always undergoes several implementation stages, namely editing, building, and launching, which may be repeated several times. Grid-based software follows the same life cycle, although there are distinct requirements and differences in the details of the stages. Different types of tools exist for assisting developers in creating innovative and bug-free software in straightforward, methodical, and efficient manners. These tools include, for example, code editors, compilers, debuggers, and profilers. As discussed in Section 2.3, these tools can be combined into one cohesive environment, an integrated development environment (IDE), that has been shown to expedite and streamline the development process. Although the tool integration results in a steep learning curve, once a particular development environment has been mastered, it can significantly increase programmers' productivity.

Grid computing enables users to build e-Science applications on distributed and heterogeneous resources that are managed by multiple organizations. These applications solve complex problems that require capacity and resources (for example, large-scale experiments, data analysis, and simulation) beyond that of individual organizations. A grid infrastructure that comprises several organizations is inherently large-scale, distributed, and heterogeneous. Grid middleware, as discussed in Section 2.6, is used to hide the heterogeneous nature and provide users and applications with a homoge-

nous and seamless environment by implementing a set of standardized and well-defined interfaces. Grid middleware services can be categorized into two types: low-level and high-level. This distinction is due to the fact that typical middleware services, whilst powerful, are too low-level for grid developers and e-Scientists to use, and thus it is necessary to provide higher level abstractions and services.

Grid application-level tools provide high-level abstractions and isolate users from the dynamics of the grid infrastructure (Section 2.7). These tools, which are built on top of the base grid infrastructure, can be employed by users to easily write and/or run grid applications. The application-level tools can be classified into three major categories: grid programming libraries; grid application execution environments; and grid integrated development environments. Grid programming libraries provide high-level functions that hide the heterogeneity and complexity of the underlying grid infrastructure from the application codes. Grid application execution environments provide software systems that simplify the execution of applications on distributed grid resources. Grid integrated development environments provide integrated tools that can be used to write and test grid applications.

The next chapter presents the research contribution of this thesis. It discusses the architecture and design of a new infrastructure that supports the development of e-Science and grid applications (ISENGARD). The infrastructure consists of novel tools that address the challenges listed in Section 1.1 and improve the process of grid software development.

Chapter 3

Architecture and Design

The first chapter of this thesis describes some challenges that make developing grid software difficult and error prone; followed by the literature review that presents a range of middleware, environments, and tools for grid application development.

We have surveyed representative examples of grid middleware in Section 2.6 and realized that they do not provide application development services for building, testing, managing, and deploying grid-enabled software. For example, there is no standard service for debugging grid applications across multiple heterogeneous resources. As introduced in Section 1.1, the lack of grid software development support is a significant barrier for e-Scientists in realizing the full potential of grid computing. In addition, the lack of such support hinders the creation of reliable and bug-free grid applications.

A survey of grid IDEs in Section 2.10 shows that they mitigate some of the issues associated with grid application development. They provide tools and functions which are specifically suited for grid computing. The majority of them are implemented by extending traditional IDEs such as Eclipse and NetBeans. As a result, these grid IDEs simplify the transition from non-grid to grid programming for developers who are already proficient with the traditional counterparts. Nevertheless, with the possibility of a large number of compute nodes being used to run grid software, it is necessary to have a facility that can automate a variety of programming activities such as

compilation, project management, and unit testing. These tasks may become tedious and error prone to perform manually as the number of target grid resources increases.

The survey of grid IDEs also shows that they are mostly monolithic development tools and they are built around a particular IDE or a particular grid middleware. Thus, programmers need to develop grid applications with a specific IDE and middleware toolkit to gain the benefit from the tools. In contrast to the approach taken by these tools, we believe that grid development functions can be generalized to a modular and integrated framework which is independent of the grid middleware and the IDE. This abstraction allows programmers to write grid applications with their preferred IDEs targeting resources that are managed by different middleware.

To address the aforementioned issues in the grid middleware and IDEs, we have developed a new infrastructure that supports the development of e-Science and grid applications (ISENGARD). The infrastructure consists of a middleware service, a modular grid development framework, and a programmable IDE framework. This chapter presents the architecture and design of ISENGARD, with particular focus on three new tools called GridDebug, Worqbench, and Remote Development Tools (RDT). First, this chapter describes the design objectives of grid application development tools and the ISENGARD approach (Section 3.1). Then, it discusses the high-level architecture of ISENGARD (Section 3.2) and the design of GridDebug (Section 3.3), Worqbench (Section 3.4), and Remote Development Tools (Section 3.5) in detail. Finally, a summary section is given at the end of the chapter (Section 3.6).

3.1 Grid Application Development

Grid application development concerns the process of writing, testing, and debugging grid-based software. It is different from the traditional application development, chiefly because of the heterogeneous and distributed nature of the target platform. Specifically, we believe that traditional programming tools do not handle the *complexity* and *scale* of the grid infrastructure

[Abramson, 2006]. In contrast to traditional systems, grid application development requires programming tools and services that are designed and engineered for grid computing [Lee et al., 2001, Bal et al., 2003, Balle and Hood, 2004]. Section 1.1 lists several specific challenges that make developing grid applications non-trivial. To meet these challenges, we believe grid development tools need to conform to a number of design objectives that are listed as follows:

- They should be scalable, meaning that they support a large number of grid resources simultaneously. Importantly, as the number of grid resources grows, the tools must allow programmers to administer, manage, and utilize the resources in an effective manner.
- They should be versatile, meaning that they operate on heterogeneous machines with a variety of hardware architectures and operating systems. Since the resources themselves may be managed by different grid middleware, the tools should not be tied to a particular middleware implementation.
- They should be extensible. Grid application development is an ongoing research area and the landscape of programming tools is evolving with a number of standards and procedures. As a result, it is desirable to have grid development tools that can be extended to operate with other software and customized to perform specialized programming tasks.
- They should be secure. Grid development tools are used in a distributed environment by multiple users, thus, the tools must preserve the confidentiality and integrity of the system and the users. However, a grid infrastructure may use resources that are distributed across multiple administrative domains. Consequently, the tools need to handle user authentication and authorization whilst complying with the local grid security policy in place.
- They should be user-friendly. Developing grid software is difficult due to the intricate nature of the target infrastructure. To compound mat-

ters, the e-Scientists may not have the technical background to understand grid programming. Thus, the tools must be as simple as possible, yet powerful enough for writing sophisticated applications. Furthermore, for obvious reasons, it is beneficial to have tools that have similar usage to traditional programming tools.

3.2 Architecture of ISENGARD

We have designed and implemented ISENGARD according to the design objectives described in the previous section. ISENGARD supports a modular collection of "pluggable" grid development tools as opposed to a single monolithic system. Specifically, the infrastructure consists of distinct software tools, modules, and plug-ins with well-defined APIs and bindings. The modular architecture and design offer a number of advantages, which are described as follows:

- ISENGARD functions are decomposed into smaller tools that can be deployed and replicated on multiple machines. This increases scalability and reliability by spreading the workload across a number of development machines and avoiding a single point of failure.
- Through the use of adapters and modules, ISENGARD works with different grid middleware supporting a variety of architectures and operating systems. It also allows ISENGARD to conform to the security framework implemented by the grid middleware.
- This modularity allows ISENGARD components to be extended and replaced as required by programmers to work with existing and new grid development tools. It also enables ISENGARD to interface with the specialized domain-specific tools required by e-Scientists.
- ISENGARD can work with a variety of clients, for example, command-line interface (CLI) programs, web browsers, or IDEs. Plug-ins for traditional IDEs can be implemented to utilize ISENGARD APIs, thus, it simplifies the transition from non-grid to grid application development.

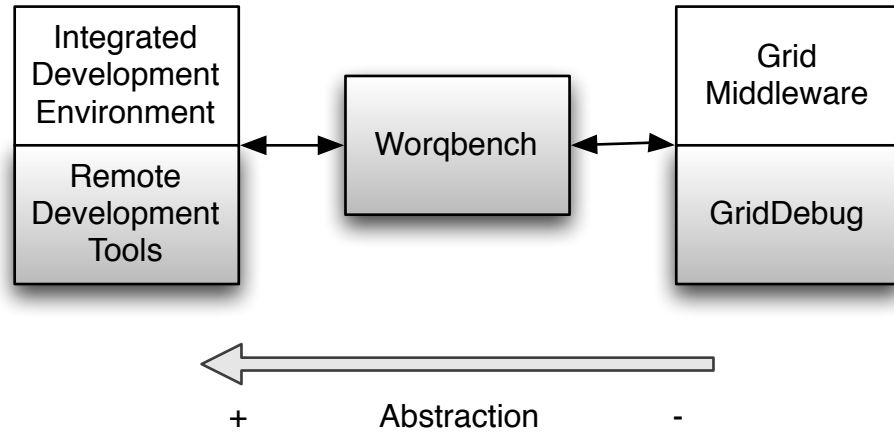


Figure 3.1: Architecture of ISENGARD

ISENGARD employs a modular and layered software architecture, which allows loose coupling between various components. It consists of three software components, namely Remote Development Tools (RDT) [Kurniawan and Abramson, 2007b, 2008], Worqbench [Kurniawan and Abramson, 2006], and GridDebug [Kurniawan and Abramson, 2007a]. These components form an integrated software infrastructure for supporting e-Science and grid application development. RDT is an IDE framework which can be incorporated into existing IDEs for developing grid applications. Worqbench is a framework that provides high-level services and API that bridge the gap between IDEs and grid middleware. GridDebug is an extension to the middleware in the form of a grid debugging service for finding and identifying software bugs in grid applications.

ISENGARD components and their relationships are illustrated in Figure 3.1, which also depicts the level of abstraction. GridDebug and Worqbench can be categorized as lower and upper middleware layers respectively (Section 2.6) and Remote Development Tools can be classified as user-level software. ISENGARD components communicate to one another through standard and well-defined interfaces. RDT utilizes Worqbench services which in turn make use of GridDebug API. In spite of that, the components are loosely coupled and their explicit interfaces enable them to be used individually. For example, a high-level software logger or tracer could be built on top of GridDebug

without being dependent on RDT or Worqbench. The next three sections describe the architecture and design of GridDebug (Section 3.3), Worqbench (Section 3.4), and Remote Development Tools (Section 3.5) in detail.

3.3 GridDebug Framework

A literature review of grid middleware in Section 2.6 shows that existing middleware doesn't address issues in debugging. Specifically, they do not provide services that support functions found in traditional debuggers, such as process control, data management, and so forth, thus it is very difficult to build a debugging tool on top of current middleware abstractions. This provides a motivation for GridDebug which is a generic debugging framework designed and built as an *extension* to grid middleware. Importantly, it conforms to the Open Grid Services Architecture (OGSA) standard (Section 2.5.3), meaning that existing middleware does not have to be replaced. The GridDebug design simplifies the process of augmenting existing middleware with a debugging service. As a middleware extension, it can leverage the security and authentication services provided by the grid middleware. Thus, GridDebug does not need to provide security-related functions such as user identification and authorization. More importantly, GridDebug does not require changes to the grid security policy in place, which is important since each grid site may have particular security requirements.

By design, GridDebug utilizes existing source-level debuggers, for example GDB [GDB, 2007], as its debug back-end. These debuggers are well-proven tools and the result of much research and development. The merits of utilizing an existing debugger far outweigh the advantages of building a new source-level debugger. The GridDebug framework encapsulates its functionality in the form of a debug library. It provides the specification and implementation of a standard set of application programming interface (API) suitable for debugging and testing computational grid applications. The API methods can be used within a grid-level debugger or other development tools that require these functions. For example, the API allows programmers to create custom tools such as high-level software tracing utilities to perform

specialized debugging tasks.

GridDebug also provides an agent-based remote debugging feature that is appropriate for grid application development. This feature allows developers to debug applications running on distributed grid resources. It has been designed to handle complex issues such as job scheduling and access restrictions across a hierarchy of resources.

The components and API of GridDebug are discussed in Sections 3.3.1 and 3.3.2 respectively. The remote debugging feature of GridDebug is described in Section 3.3.3 while the design conclusion is given in Section 3.3.4.

3.3.1 GridDebug Components

GridDebug is not designed as a stand-alone monolithic program but rather as component-based software, which simplifies adding its services to existing grid middleware. Specifically, it consists of four main components, which include an interface to grid middleware, a debug back-end wrapper, a debug client, and the debug library. Figure 3.2 shows the components and the overall architecture in relation to the grid middleware.

The following four sections describe the components, namely the debug client (Section 3.3.1.1), the middleware compatibility layer (Section 3.3.1.2), the debug API library (Section 3.3.1.3), and the debug back-end (Section 3.3.1.4).

3.3.1.1 Debug Client

A developer utilizes the debugging service by using a debug client. The client can be written and customized to accommodate different developer requirements and interactions. The client can utilize various methods in the debug API library. In addition, it can also take advantage of services offered by the middleware such as security and notification services.

The debug client can be implemented as a simple command-line interface (CLI) program similar to a traditional CLI debugger or as a plug-in of an IDE such as NetBeans (Section 2.3.5) or Eclipse (Section 2.3.6). Furthermore, developers can extend or write sophisticated debug clients, for

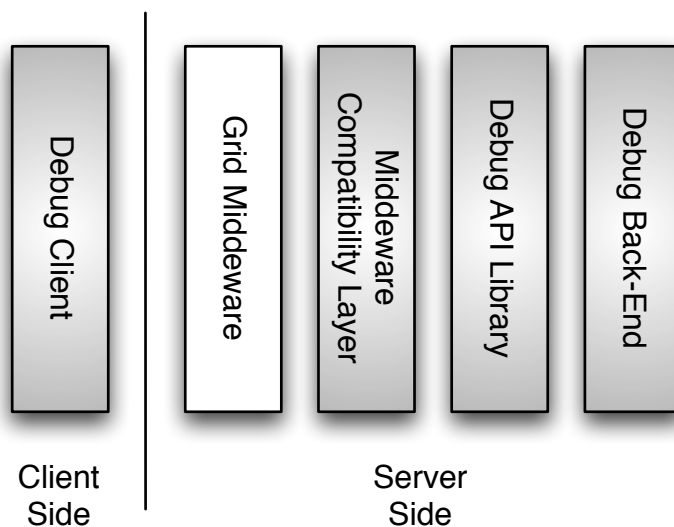


Figure 3.2: GridDebug architecture

example, high-level tracers and tools for enforcing software contracts (pre and post conditions) [Meyer, 1991].

3.3.1.2 Middleware Compatibility Layer

GridDebug encapsulates a *generic* debug library as a *middleware-specific* extension. This encapsulation is provided by the middleware compatibility layer. It is essentially a grid middleware plug-in that wraps and exposes the debug library as a grid debugging service. This component acts as a translator between middleware-specific and generic debug interface. The layer is written according to the extension mechanism of the middleware. For example, with OGSA-based middleware, the compatibility layer is implemented as a grid service that needs to be deployed into a service container.

3.3.1.3 Debug API Library

The main component of the GridDebug framework is a debug library with a well-defined debug API. The library provides an abstraction of various debugging operations and it has been devised to be modular and independent of the grid middleware and the debug back-end. This design provides a

solution to the lack of standards for debugging in the domain of grid software development. The debug library could be adopted as a standard of grid debugging API.

The API defines a generic debug model and a collection of methods and objects for grid application debugging. The API includes, for example, methods to define process sets, to create/delete breakpoints, and to control the execution of programs. The API is discussed in more detail in Section 3.3.2.

3.3.1.4 Debug Back-End

GridDebug leverages well-proven traditional debuggers as its debug back-end. These debuggers, for example GDB [GDB, 2007], are widely used development tools and they work with a variety of computer architectures and operating systems. The debug back-end is the component that performs the actual low-level debug operations as defined by the debug API. The API itself is generic and independent of the debug back-end. Thus, a different source-level debugger, for example Intel Debugger [IDB, 2008], can be utilized by implementing an interface to the debug library, without changing the API method signatures.

3.3.2 GridDebug Application Programming Interface

The GridDebug API specifies a debug model, methods, and data structures using an object-oriented paradigm. Classes are defined to represent entities such as processes, breakpoints, and events. This object-oriented approach has several advantages such as modularity, encapsulation, and abstraction [Gamma et al., 1994]. In addition, through class inheritance, the debug model and API can be extended to provide more specialized functions.

The GridDebug API is accessed through a `DebugSession` object which represents a single debugging session for a user. It simplifies API access by encompassing all debugging activities for an application into a single addressable entity. A `DebugSession` object comprises a number of smaller and specialized objects. Specifically, they are employed for event notification (`DebugEventManager`), for storing configuration parameters (`DebugConfig`), for

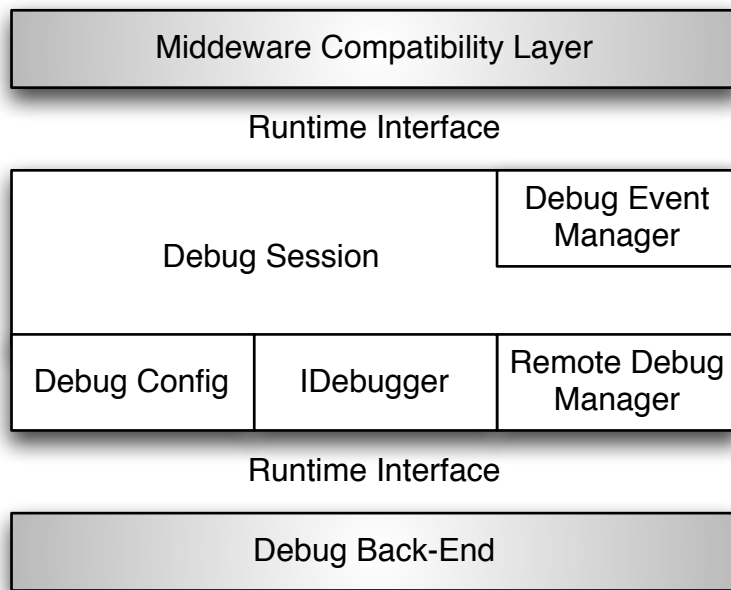


Figure 3.3: Components of the debug API library

remote debugging (`RemoteDebugManager`), and for accessing the underlying back-end debugger (`IDebugger`). The components of the debug API library are depicted conceptually in Figure 3.3 and they are described as follows:

- `DebugEventManager`. In response to program errors or events, a debugger may issue asynchronous messages, such as when a breakpoint event or a program runtime error occurs. To handle these messages, the GridDebug API employs a class, `DebugEventManager`, which provides an event queue with appropriate retrieval methods. The debug client needs to check the event queue at regular intervals to retrieve debug messages and output; and responds accordingly. Alternatively, if the grid middleware supports a notification service, the client can be notified of any messages from the debug library. This event-based mechanism leads to an abstract and minimal coupling between inter-dependent components [Riehle, 1996]. The event producers (debug back-ends) need not assume anything about the event observers (debug clients).

- **DebugConfig.** The GridDebug API provides a class, `DebugConfig`, to store debugger configuration parameters. It allows developers to query or alter the general behaviour of the API to accommodate different debugging task requirements. The configuration operations can be performed by invoking various `DebugConfig` methods.
- **RemoteDebugManager.** Remote debugging is a necessary feature for grid application development due to the distributed nature of grid infrastructure. The GridDebug API employs a class, `RemoteDebugManager`, as an abstraction layer that manages and encapsulates all network-related operations. It relieves the debug client from the low-level details of the communication protocols. This feature allows a remote debugger such as GDB/GDBServer [GDB, 2007] to be utilized as the back-end debug engine. Section 3.3.3 discusses the GridDebug remote debugging mechanism in greater detail.
- **IDebugger.** The interface between the debug library and the back-end debugger is provided by a class, `IDebugger`. It encapsulates the underlying debugger operations into a set of GridDebug API methods. This abstraction class contributes to the modular design of the GridDebug framework. A new back-end debugger can be utilized by extending and modifying the `IDebugger` class without altering the rest of the GridDebug API library.

3.3.3 Grid Remote Debugging

Remote debugging is the process of debugging an application running on a separate system different from the debugger's system. Generally, the debugger initiates the communication with the debugged application using a network connection such as a TCP/IP socket. However, this technique is not suitable for grid remote debugging since it does not address issues such as job scheduling and access restrictions across a hierarchy of resources.

In a typical grid environment, program execution is managed by a local queue manager (for example, PBS [PBS, 2008], LSF [LSF, 2008], and SGE

[SGE, 2008]), which schedules jobs according to criteria such as resource availability and processor load. This batch processing scenario makes it difficult to debug an application interactively because the time when the scheduler actually starts program execution cannot be easily determined. As a result, it is necessary to employ ad-hoc techniques such as polling the scheduler regularly to check whether the application has been started, and then manually attaching a debugger to the process. In a grid, this process might need to be repeated across multiple resources, possibly using different local schedulers, making the technique cumbersome and error prone.

In addition to the job scheduling issue, there is another complicating matter. Many grid testbeds are actually built from distributed clusters, consisting of a front-end machine and multiple back-end processor nodes. These back-end nodes are usually only accessible from the front-end, which may in turn be behind a gateway server or a firewall. Thus, debugging an application running on the execution nodes of a cluster may require access to a hierarchy of intermediate computers. However, depending on the grid security policy in place, a grid user may not have direct access to all of these machines, again complicating debugging.

As a solution to address these issues, GridDebug utilizes an agent-based callback mechanism for grid remote debugging. GridDebug employs a debug agent that prepares an application for remote debugging and contacts the debug back-end. In this mechanism, it is the debugger that waits for a callback connection from the debugged application.

This technique alleviates the need to constantly poll the scheduler to check whether an application has been started. Furthermore, in a grid environment with a hierarchy of resources, a programmer does not need to have access to intermediate machines, provided that the back-end node where the GridDebug agent is running can access the GridDebug service. This technique also adds another layer of security since the debugging activity is initiated by the application rather than by the programmer, thus a user cannot debug another user's processes. An illustration of the GridDebug remote debugging technique is depicted in Figure 3.4.

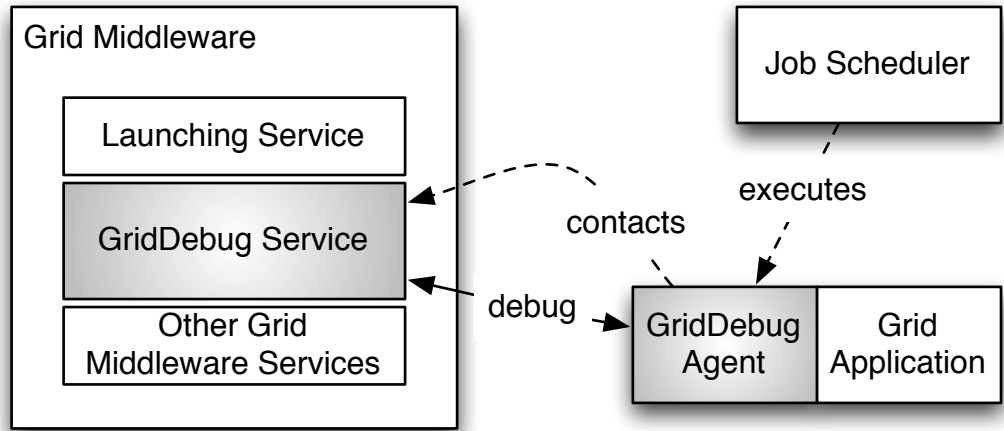


Figure 3.4: Grid remote debugging

3.3.4 GridDebug Design Conclusion

Rather than developing, yet again, a debugger for a particular grid middleware, we have designed and implemented GridDebug as a modular framework that can accommodate different grid middleware and different debug back-ends. This design simplifies extending existing middleware with a debugging service and leverages existing debuggers. GridDebug utilizes the security and authentication services provided by grid middleware and it does not require changes to the local grid security policy. The lack of grid debugging standards is addressed by proposing and implementing generic methods and API that could support a number of tools and middleware for grid application debugging.

The GridDebug framework and its functionality can be utilized and extended by other tools. For example, profilers and high-level tracers can be built on top the framework. In addition, customized debugging tools for a programmer's particular need could be implemented with ease, by leveraging the GridDebug API.

GridDebug provides an agent-based remote debugging with a callback mechanism. By implementing a callback mechanism in the service, the task of initiating a debugging session is inherent in the way an application is initiated. This mechanism alleviates the need for a programmer to constantly

poll the job scheduler. The callback mechanism also eliminates the need for the programmer to access intermediate resources (firewall server, gateway machine, or front node) to debug an application. The programmer is required, however, to wait for an incoming connection from the execution node where the debugged application is running and this is managed by the GridDebug framework. In addition, since the debugging activity is initiated by the application rather than by the programmer, a user cannot arbitrarily debug another user's processes.

3.4 Worqbench

Worqbench is an integrated and modular framework that links IDEs with grid middleware for e-Science application development. It provides various services (packaging, launching, and editing) that correspond to the stages in the implementation phase of a software development life cycle (Section 2.1). Worqbench provides an infrastructure that isolates the development services from both the grid middleware and the IDEs. As a result, the system becomes very generic, and its adaptation to particular client scenarios or requirements only requires the creation of appropriate customized modules.

Worqbench consists of core components and a set of modules, or plug-ins, that implement the interfaces to various middleware and IDEs. The framework links an IDE to grid middleware in a three-tier model as depicted in Figure 3.5. This three-tier model ensures flexibility and modularity. It allows the aggregation of grid resources with different architectures and middleware into a single grid environment or testbed accessible from the programmer's preferred IDE. The left tier supports the user interface, the IDE and associated plug-ins. The middle tier implements the framework that coordinates the connection between the IDE and the grid middleware. The right tier provides the underlying back-end structure that may consist of one or more resources managed by different grid middleware.

The Worqbench design is generic and it can accommodate and support other client applications besides IDEs, as long as they implement the necessary communication interface. This opens up possibilities for a wide range of

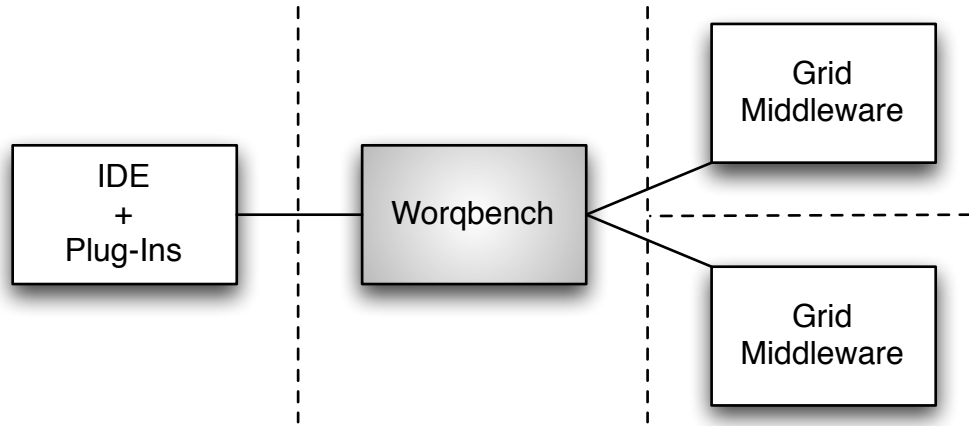


Figure 3.5: Three-tier relationship model

user interactions with Worqbench. For example, a web browser with a Java applet or an Ajax user interface library [Garrett, 2005] can be used to access Worqbench. This provides an accessible and ubiquitous platform similar to a grid portal (Section 2.9.1) for developing grid software. Alternatively, a program with a command-line interface can also be implemented to utilize Worqbench services, delivering a suite of small specific utilities comparable to a Unix shell. This shell could provide a versatile and powerful environment for grid programmers.

Further discussion of Worqbench is given in the following three sections. The components and the system model are described in detail in Sections 3.4.1 and 3.4.2 respectively; while the design conclusion is presented in Section 3.4.3.

3.4.1 Worqbench Components

The design objective of Worqbench is to provide an extensible and generic framework, that can work with various grid middleware and IDEs. This necessitates a component-based software architecture. Worqbench is decomposed into a number of smaller components that result in more manageable software where each part can be replaced with minimal effect to the system. Specifically, the components are: connection interface (Section 3.4.1.1),

code repository (Section 3.4.1.2), development manager (Section 3.4.1.3), database (Section 3.4.1.4), development services (Section 3.4.1.5), and middleware adapter (Section 3.4.1.6). These components are depicted in Figure 3.6.

3.4.1.1 Connection Interface

The connection interface handles the connection from the IDE and other clients to the framework. Each interface handles one particular type of communication protocol, for example, SOAP [W3C, 2007a] or XML-RPC [Winer, 1999]. In addition, this component also performs the authentication and authorization of Worqbench users based on the security credentials stored in the central Worqbench database.

The connection interface provides a clean and well-defined abstraction layer that allows Worqbench to support various IDEs and clients. If an IDE requires a specific communication handler, a suitable connection interface can be written for that particular IDE without changing the other components of Worqbench.

3.4.1.2 Code Repository

Worqbench provides various development functions that are grouped into three services: packaging, launching, and editing. These functions, for example, compiling and deploying an application on grid resources, require access to the application's project. A project is defined as a grouping of all the files necessary to build an application, such as library, header, and source files. Worqbench employs a code repository to manage and store the contents and metadata details of application projects. It also acts as a version control system that can save multiple project revisions. The code repository enables Worqbench services to perform "offline" or unattended operations with the projects on users' behalf. For example, executing automatic builds or test harnesses.

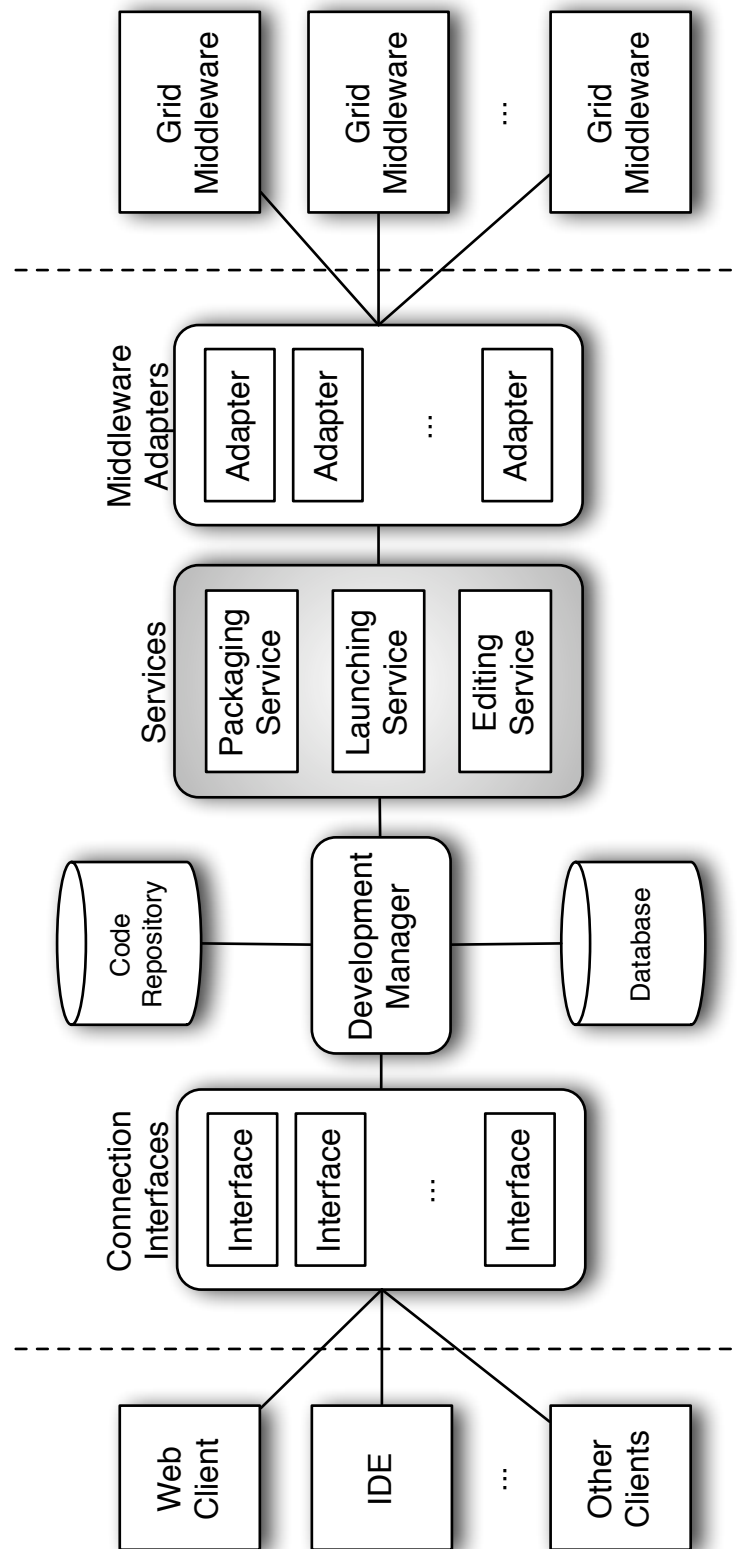


Figure 3.6: Key components in Worqbench

3.4.1.3 Development Manager

The development manager coordinates the interactions of various Worqbench components. It specifies bindings and mechanisms that allow the components to communicate with one another. Thus, it allows a component to be replaced or extended, provided that it adheres to the specified interface.

The development manager is the application logic component of Worqbench and it is also responsible for managing various entities such as users, projects, resources, and resource groups. The relationships between entities are defined by a Worqbench system model, which is discussed in Section 3.4.2. This system model is maintained and stored by the manager in the database.

3.4.1.4 Database

Worqbench requires a storage server to save the system state. This function is provided by the database component that stores information, for example, user passwords and security certificates for accessing grid resources; projects and sessions configurations; and details of application executions. The database component is loosely coupled to the framework and can be deployed on another machine to lessen the load on the Worqbench server. Using an external and separate database server is advantageous. For example, multiple front-end servers can be configured to utilize a central database to improve the scalability and reliability of Worqbench. Furthermore, the database itself can be clustered and replicated to avoid the loss of data and increase the robustness of the overall system.

3.4.1.5 Development Services

Worqbench provides editing, packaging, and launching services that correspond to the stages in the implementation phase of a software development life cycle (Section 2.1). The services offer generic APIs and methods, that are isolated and independent of the grid middleware. These generic functions are mapped to specific middleware APIs or commands by the middleware adapter

component. As a result, Worqbench clients only need to target one set of interfaces to work with various middleware stacks.

The editing service provides the management of a software project, including its multiple revisions on a local machine. The service is closely integrated with the code repository. It allows a user to retrieve and query project files. A user can check out a local working copy of the project, perform the necessary editing, and commit the code to the Worqbench code repository.

The packaging service is responsible for building or compiling a grid application on remote resources. Typically, it does this by transferring a compressed file containing source code data to remote machines; setting up and initializing the appropriate directories; and executing a build script to compile the application. The service assumes that the target resources have a suitable environment, including development tools and libraries, to build the application.

The launching service manages the execution of an application on grid resources. The service supports two modes of execution: normal and debug. The normal mode utilizes the underlying grid middleware job submission tools whilst the debug mode utilizes the GridDebug debugging service.

Worqbench services augment existing grid middleware by providing specific software development functions that are currently not available. They are generic and middleware-independent, providing the necessary support for grid software throughout its life cycle.

3.4.1.6 Middleware Adapter

The middleware adapter is the glue between the Worqbench framework and the low-level grid middleware. It translates generic input commands from the framework to specific middleware commands or APIs. Each adapter handles one particular grid middleware platform and it uses the authentication information provided by the user such as X.509 proxy certificates or user passwords to gain access to the grid resources.

This component hides particular middleware details from Worqbench services, and it allows the framework to support various existing and, more

importantly, future grid middleware. If a programmer uses resources handled by a new grid middleware, a suitable adapter can be written for that particular middleware without altering the other Worqbench components.

3.4.2 System Model

There are a number of participating "objects" or entities involved in grid application development. For example, e-Scientists or programmers; target grid resources; and application projects. The relationships between these entities must be properly defined and understood. The aim of this is to allow greater interoperability between Worqbench components or services that work with the entities.

Worqbench maintains an internal model of the system and stores the state of various entities such as users, projects, resources, and resource sets. A conceptual entity-relationship diagram of the model is shown in Figure 3.7. The diagram uses Barker's Notation [Barker, 1990] to represent relationships between entities with connecting lines and symbols that indicate the cardinality of the relationship. Here, a dash represents *one* and a crow's foot represents *many*. For example, the figure shows that a user can have many references to projects but a project is owned or can only be referenced by one user. We believe that the Worqbench system model depicted in Figure 3.7 is representative of common grid application development scenarios. There are seven model entities that are described below, followed by the discussion on the Worqbench system model.

- **Resource.** It represents a distinct machine or resource in the framework. It keeps general and system-specific information of a resource, for example, host name, machine type, network address, and resource description.
- **User.** The system model revolves around a User entity. It represents one distinct user of the Worqbench framework. The entity stores the user details and the Worqbench authentication information.

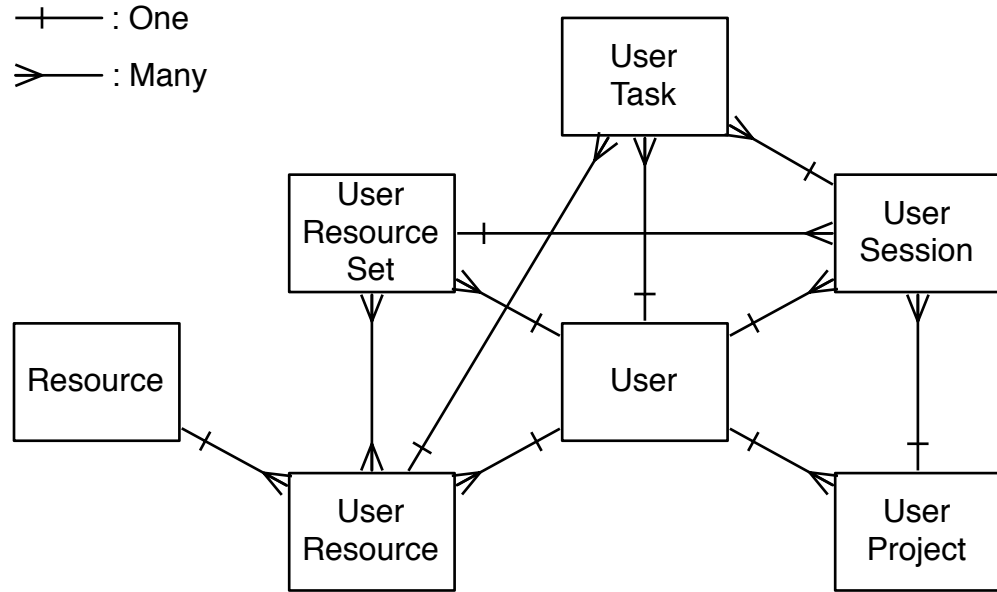


Figure 3.7: System model of Worqbench

- **UserResource**. This entity represents a machine or resource for a specific user. It keeps a reference to a **Resource** entity and stores user-specific information such as the account user name for accessing the machine and the corresponding password credential.
- **UserProject**. The application development project for a user is represented by the **UserProject** entity. It stores a data object containing all of the necessary files for a development project such as header, source code, and image files. The model allows a user to maintain multiple independent projects.
- **UserResourceSet**. Grid users may want to group some resources to form a collection or a resource set. This set is represented by the **UserResourceSet** entity. It maintains a collection of membership references to various **UserResource** entities.
- **UserSession**. A **UserSession** entity represents a development session for a user. It is defined as a programming session where a user develops an application in one **UserProject** to be tested and executed on one

UserResourceSet.

- UserTask. It represents a remote operation invoked with a UserSession entity. For example, in a particular development session, a user invokes a software build or compilation. This operation is represented by a UserTask entity that can be inspected by the user for its attributes such as execution status and output.

The Worqbench framework distinguishes between system and user resources: the Resource and UserResource entities respectively. This distinction allows the framework to avoid repetition when performing system centric operations (regardless of the resource users), such as testing for network connectivity or replacing the SSH host certificates.

The UserProject entity allows grid programmers and e-Scientists to group related development files into one self-contained data object. This simplifies the management of source files and their deployment on grid resources. Additionally, resources can be grouped together into a set (the UserResourceSet entity). It allows a Worqbench user to invoke a development function on a set and let the framework perform the actual operations on individual resources. This mechanism streamlines application development on multiple grid resources.

The process of developing a grid software project on a resource set is encapsulated by the UserSession entity. It holds references to UserProject and UserResourceSet entities. The entity provides a clear and definite linkage between the software and the target platform; and allows Worqbench services to perform operations on users' behalf unambiguously. Various methods of Worqbench development services (Section 3.4.1.5) work with a UserSession entity and they operate using its project and its resource set. The invocation of a Worqbench service method is represented by the UserTask entity, which stores various information such as the time, status, output, and target destination of the execution. It allows users to inspect deferred or long-running grid operations at a later time.

3.4.3 Worqbench Design Conclusion

Worqbench allows e-Scientists and programmers to aggregate grid resources with different architectures and middleware into a single grid environment or testbed accessible from the users' preferred client or IDE. It provides common services that support the software development life cycle of grid applications. Worqbench is modular and extensible; and it addresses the challenges listed in Section 3.1.

Worqbench scales the grid development process to large environments because it performs development tasks on behalf of a user rather than requiring the user to perform these manually. For example, the packaging service copies files from the user's project and builds the project on each grid resource without user interaction. This helps the user manage a software development project across a large number of heterogeneous target resources, as experienced in the grid.

The Worqbench framework is versatile in accommodating resources managed by a variety of grid middleware. Furthermore, it is generic and it can be accessed by IDEs and other client applications. For example, customized programming utilities to suit particular requirements or user-friendly web browsers.

Worqbench exposes heterogeneity to the user in a controlled way and therefore helps manage the challenge in the grid. For example, the development manager can check whether a group of resources is sufficiently homogeneous, and if they are similar, it can present them as a single resource type. This helps the user manage the variability in the grid infrastructure.

Worqbench provides unified services for packaging, launching, and editing grid software. It proposes and implements standard methods that could support a number of tools and projects for grid application development. These services could be adopted as open standards of grid development services that can work with various middleware stacks.

3.5 Remote Development Tools

Remote Development Tools (RDT) are a set of modules and a user-level framework that augment an IDE for grid application development. Whilst, traditional IDEs provide sufficient programming tools for building grid software, they are often designed and implemented for a single platform, namely, the local machine where the IDEs run. Thus, various programming tasks such as compilation, project management, and unit testing; may become *cumbersome* and *complicated* for grid infrastructure that contains a large number of different resource types. RDT offers new features that simplify and improve various IDE functions for grid application development.

RDT provides a runtime programmable framework with a system model, API, and an event mechanism. It allows developers to *automate* a variety of programming activities that become tedious if performed manually. In addition, RDT also provides a multi-view code editor and a remote launching mechanism, that extend the IDE support for software development on distributed and heterogeneous grid resources. The editor can be used to write architecture-specific code without being hindered by the difficulty in managing code variations. The remote launching mechanism allows application execution on multiple remote targets simultaneously and transparently.

RDT is not a replacement for an integrated development environment, rather, it is designed as an extension to an IDE. It utilizes and extends standard functions of the host IDE such as integrated code editing, project management, file/folder browsing, and so forth. It leverages GridDebug and Worqbench services for the low-level grid development functions. RDT has a high-level, IDE-independent, design and architecture. As a result, it can be integrated with modern extensible IDEs such as NetBeans (Section 2.3.5) or Eclipse (Section 2.3.6).

The architecture of RDT is depicted in Figure 3.8. It consists of two software layers: user interface (UI) and non user interface layers. The layer division ensures modularity and simplifies the integration of RDT into an IDE [Wasserman, 1989]. The user interface layer can be customized according to the base IDE whilst retaining the principal non-UI part. The non-UI layer

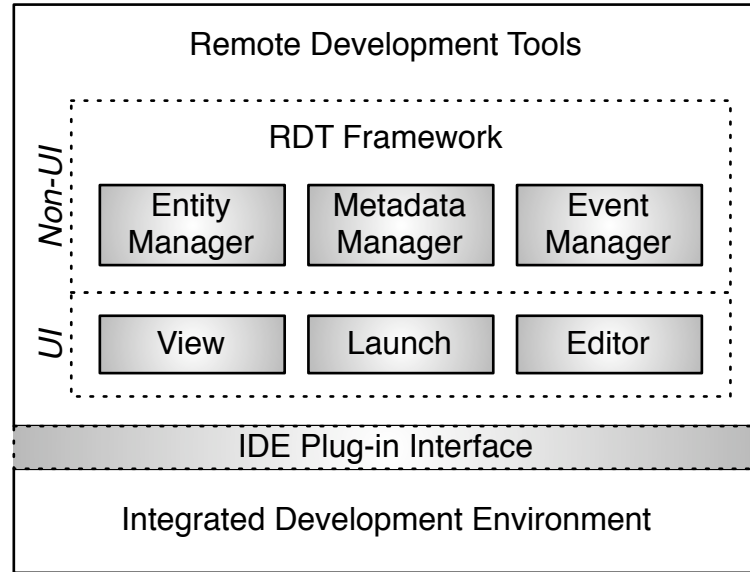


Figure 3.8: RDT architecture overview

forms the RDT framework with APIs and a set of managers whilst the UI layer provides novel GUI tools for grid software development.

Further discussion of RDT is given in the following three sections. The RDT framework and user interface are described in detail in Sections 3.5.1 and 3.5.2 respectively. The RDT design conclusion is presented in Section 3.5.3.

3.5.1 RDT Framework

Traditional IDEs are geared towards local application development and normally target the platform on which they are installed. Thus, IDEs are mainly concerned with software projects, folders, and files as opposed to target resources. As a result, the process of developing, executing, and debugging grid applications over *a large number of heterogeneous resources* becomes inefficient. To alleviate this problem, RDT introduces several improvements.

First, RDT extends the conventional IDE model to include grid resources and resource sets as *first-class objects* similar to files, folders, and projects in the IDE. It allows developers to inspect and manage the target platforms

from the IDE. More importantly, the model ties the application development process to the composition of the grid environment itself. It allows RDT to recognize the relationship that exists between software projects and various grid resources; and perform operations or automation scripts based on this association. As a result, the grid software development process is streamlined since repetitive management functions can be handled by RDT on a user's behalf.

Second, RDT provides a *runtime programmable facility* that allows the IDE to be scripted during its execution. Modern IDEs often provide extension points and mechanism that allow them to be extended through plug-ins or enhancements. However, these plug-ins are normally static and are loaded when the IDE starts execution. In contrast, RDT allows developers to call or change the functionality of the grid IDE during *runtime*. Thus, in addition to the common point-and-click interaction, developers can write scripts to interact and *drive* the IDE. It provides versatile and powerful means to utilize the IDE for grid application development.

Whilst these ideas are valuable even for conventional software development, we believe they are particularly effective for grid software development, because they automate processes that would not scale as the number of target platforms increases. The rest of this section describes the framework components in detail as follows:

- Section 3.5.1.1 presents the RDT entity manager and system model based on the Worqbench design with an abstraction of resources and sets as first-class objects.
- Section 3.5.1.2 discusses a system-wide tagging system for attaching metadata information to various entities such as resources and projects.
- Section 3.5.1.3 describes an event notification system with user-defined callback methods.
- Section 3.5.1.4 discusses the application programming interface that allows RDT and the IDE to be accessed programmatically during runtime.

3.5.1.1 Entity Manager

The RDT framework maintains an extended IDE model that includes grid resources and resource sets as first-class objects. This model is based on the Worqbench design (Section 3.4.2) since it is comprehensive and adequate to represent common grid application development scenarios in the IDE. Specifically, the RDT model consists of entities such as resources, resource sets, projects, sessions, and tasks.

The administration of RDT entities are handled by various managers, namely, the Resource Manager, Set Manager, Project Manager, Session Manager, and Task Manager; which keep references to all entities in the system. Managers provide RDT users with *single access points* that simplify the management and coordination of entities across the system. They offer methods that let programmers create, delete, query, and access the entities programmatically. These managers communicate with various Worqbench services and act as translators between Worqbench and RDT system models.

3.5.1.2 Metadata Manager

The RDT system model allows the framework to recognize the relationship between various entities. However, it does not specify the semantics of the relationship. As a solution, RDT employs a tagging system and its corresponding manager that allows metadata information to be attached to RDT entities. A tag is a relevant keyword or term associated with or assigned to an entity, thus describing the item and enabling keyword-based classification of information [Marlow et al., 2006]. For example, a Linux machine can be tagged with `i686 kernel-2.6.20 smp development-machine condorpool` or a Sun workstation can be tagged with `sun sparc solaris globus`.

This system-wide tagging system facilitates the sharing of metadata between various entities; and enables the framework to understand relationship semantics. As a result, it can perform development tasks on a user's behalf based on the semantic information. For example, RDT can be instructed to automatically group or add all resources tagged with `ubuntu` or `redhat` into a `linux` resource set. This automation can lead to an increase in programmers'

productivity.

3.5.1.3 Event Manager

IDEs employ an event mechanism for certain functions. For example, the editor can be instructed to highlight a particular line of code when a breakpoint-hit event occurs or the compiler can be set to build a project when a source file is saved.

Similarly, RDT provides an event manager based on the framework system model with user-defined callback methods. This mechanism offers automations that can lead to improvements in the process of grid application development. For example, a user can write custom methods that perform various checks and operations when specific events occur. As an illustration: a warning can be displayed when a grid resource tagged with `windows` is added to a `unix` resource set; source files that need to be updated are highlighted when a new grid resource is added to an `experimental test bed` set; and so forth. This simple yet powerful mechanism increases the efficiency and effectiveness of using an IDE for developing grid software.

3.5.1.4 RDT Application Programming Interface

The RDT framework defines a set of API methods that allow programmers to write scripts which utilize RDT entities and perform various IDE functions. It offers a collection of methods that provides support for all common operations to create, delete, query, and access RDT entities. In addition, it also offers methods to invoke and call a variety of IDE functions such as file manipulation, project management, and application launching.

RDT scripts can be coded to automate and repeat many tedious grid programming tasks, such as software compilation and unit testing, across a variety of resources. Thus, this automation simplifies grid application development for a large number of target platforms efficiently, a process which is significantly more complex than when software is developed for a single platform. Moreover, scripting allows the user to tailor these functions for variations in particular grid infrastructure. Additionally, there are other

advantages of using a high-level scripting language such as late binding, reflection support, expressive syntax, and dynamic typing [Ousterhout, 1998, Prechelt, 2003].

3.5.2 RDT User Interface

In addition to the framework, RDT also includes GUI components in the IDE for grid application development. These components provide management functions and viewers for RDT entities; a remote launching facility; and a novel multi-view code editor. They are presented in further detail as follows:

- Section 3.5.2.1 presents RDT entity viewers for managing entities such as resources and resource sets.
- Section 3.5.2.2 describes a launching support to handle application execution and debugging on multiple remote machines.
- Section 3.5.2.3 describes a multi-view code editor that displays source code based on user-given keywords or tags.

3.5.2.1 RDT Entity Viewer

The management of various entities is provisioned by the RDT entity viewers that provide IDE components for showing and administering, for example, grid resources, resource sets, and development tasks. Viewers take advantage of various high-level GUI widgets supplied by the base IDE. The entity viewers simplify the management of a large number of heterogeneous grid resources and sets for e-Scientists.

3.5.2.2 Remote Launching Support

A conventional IDE normally has a single machine perspective, where the launching machine is the same as the development one. A launching machine can be defined as the node on which applications are being executed or debugged. A development machine is the computer that runs the IDE and acts as an interface to programmers. This single machine perspective

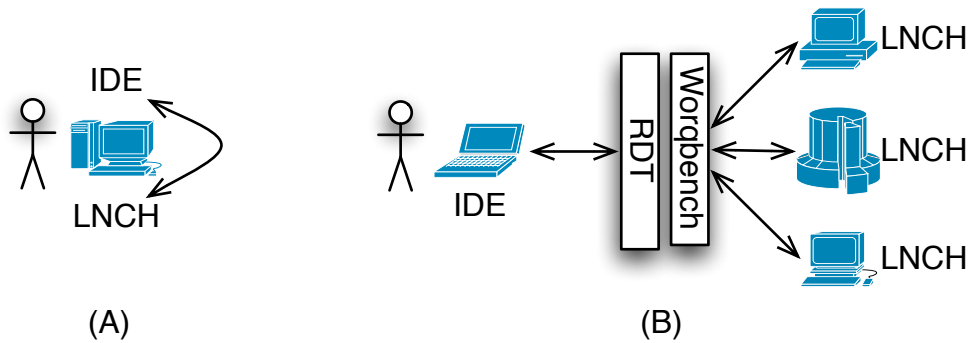


Figure 3.9: Interactions between development and launching machines

is inadequate in the grid where multiple machines may be involved in the development process. RDT reduces the coupling by following a model where the launching machines are remote resources connected to the IDE over a network. It allows grid programmers to handle multiple remote targets simultaneously. Figure 3.9 illustrates two conceptual interactions between the development (IDE) and launching (LNCH) machines. In Figure 3.9, (A) shows a tight coupling of the traditional single machine perspective; while (B) depicts the remote launching support with multiple machines.

RDT extends the launching mechanism of the IDE to handle multiple remote machines. It leverages Worqbench services and together they act as a proxy for these machines. They transparently copy source code files from the IDE to the launching nodes and build the applications. They also transparently pass input/output of the compilation, execution, and debugging operations. The remote launching support of RDT allows grid programmers to compile, run, and test applications on a number of heterogeneous resources at the same time. This mechanism is more efficient compared to manually transferring the source files and executing the applications on each individual resource. More importantly, the whole process is transparent and seamless to the programmers.

3.5.2.3 Multi-View Code Editor

Writing grid applications for a highly heterogeneous and distributed infrastructure is complex and error prone. The infrastructure may introduce vari-

ations in source code due to resource heterogeneity. Further complicating matters, programmers may need to manually manage these code variations. Different operating systems and computer architectures may provide different function calls, arguments, and interfaces. Some of these concerns are alleviated by the use of grid programming libraries, for example SAGA [Goodale et al., 2006, Jha et al., 2007, SAGA-RG, 2008], or cross-platform abstraction libraries such as Apache Portable Runtime [APR, 2008], GLib [GLib, 2008], or Netscape Portable Runtime [NSPR, 2008]. However, these libraries may not cover all specific instances such as accessing low-level processor instruction sets to speed up computations, or utilizing specialized operating system interfaces.

The typical method to write multi-platform code is to put the variations in the source code itself by employing pre-processor and macro directives, e.g. `#define` and `#ifdef` [Aversano et al., 2002]. Macro and pre-processor directives are difficult to manage and error-prone [Ernst et al., 2002]. For example, adding a new unique platform requires the programmer to find, inspect, and possibly change every occurrence of the directives. In addition, directives also make understanding the source files difficult because of the additional code variations [Ernst et al., 2002]. Various tools and editors have been developed in the software engineering domain to manage such directives [Kruskal, 2000, Livadas and Small, 1994, Kullbach and Riedinger, 2001].

We have designed and developed a multi-view code editor in RDT that allows grid developers to write architecture-specific code without being hindered by the difficulty in managing code variations. The main idea of the editor is to separate the display of source code from the textual representation. Figure 3.10 shows the difference between a standard (A) and a multi-view (B) code editor. Unlike a standard text editor that normally displays all contents from a file, the multi-view editor only shows parts of the file that are relevant to a certain context and hides the irrelevant parts. By changing the context, a different view of the file is shown. The context is specified through user-given keywords or tags. The concept of tagging is a familiar idea in the domain of software development. Conditional compilations and `#ifdef` which are the traditional ways to mark portions of code with metadata information, can

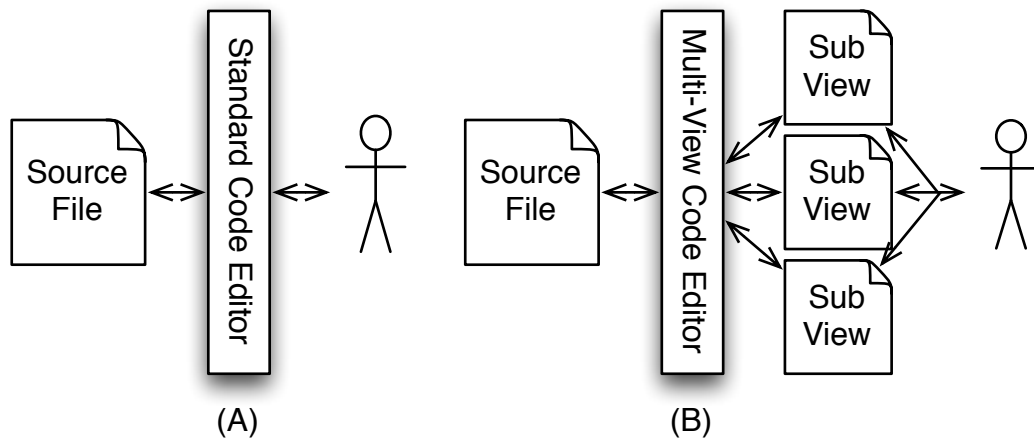


Figure 3.10: Standard and multi-view code editors

be considered as tagging systems. For example in `#ifdef WIN32`, the keyword `WIN32` can be regarded as a tag, thus, tagging is a natural and familiar concept to application developers. The multi-view code editor utilizes the RDT metadata mechanism. As a result, the source code files share the same tagging system with RDT entities such as projects and resources. It allows the framework to recognize the relationship that exists between the source files and various grid resources; and perform user-defined operations based on this association. It can automate and simplify the process of writing grid applications for multiple architectures and operating systems.

Figure 3.11 depicts a conceptual usage scenario and the design of the multi-view editor. The Document Partitioner splits a source file up into several text sections according to user-defined partitioning rules. The text sections, which represent code variations in the source file, are labeled or annotated with tags. A user specifies a new sub-view by providing the editor with a list of tags. The Document Renderer then presents the sub-view that shows text sections that match the user-given tags. The editor does not add extra information to the files nor modify them with the exception of the editing performed by the user. As a result, the user is not tied to the multi-view editor and can still use other tools to edit the files.

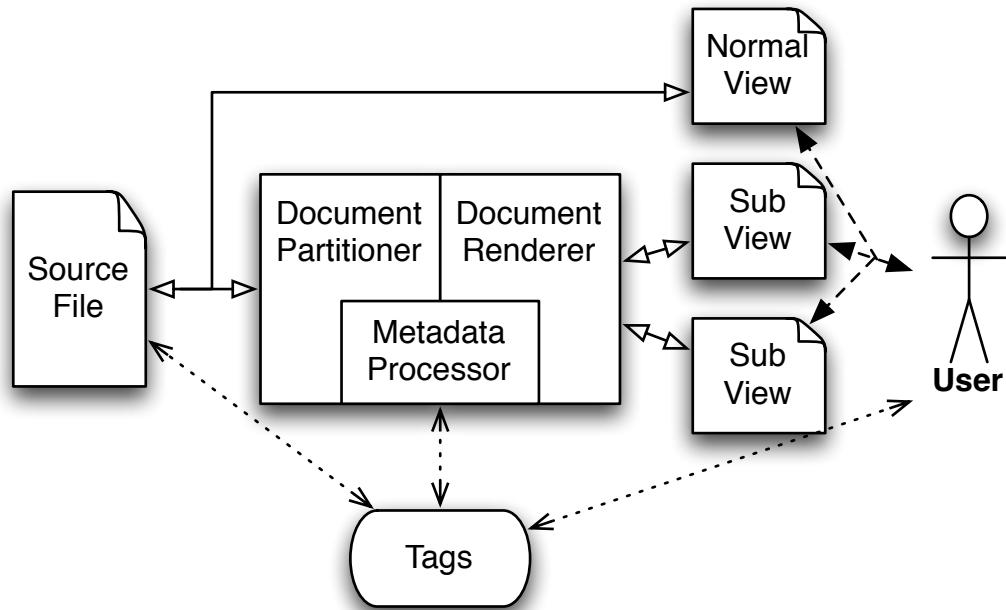


Figure 3.11: Design of multi-view code editor

3.5.3 RDT Design Conclusion

Different to other grid IDEs mentioned in Section 2.10, RDT implements a user-level IDE framework for grid application development that can be incorporated into traditional IDEs. The framework allows programmers to write scripts that utilize RDT and IDE APIs to call or change the functionality of the IDE during its execution.

The RDT system model defines an abstraction of resources and sets as first-class objects. It enables the IDE to recognize the relationship that exists between software projects and various grid resources. The semantics of the relationship can be specified through a system-wide tagging mechanism. In addition, RDT employs an event mechanism that allows programmable actions to be invoked based on certain conditions or events.

RDT improves the process of grid software development by automating many tedious grid programming tasks, particularly when a large number of resources are involved. It opens up possibilities for various interactions between application source code and target resources. For example: high-

lighting source files that need to be updated when a new resource is added to the test bed, by looking at the resource tags; listing code portions that will not be executed in selected computers, by examining the resource and code tags; and so forth. In addition, the RDT framework allows the IDE to be driven by user-defined scripts. The use of a high-level scripting language is advantageous since the user can customize the tool functions to cater for variations in their particular grid infrastructure.

RDT offers a multi-view code editor that allows programmers to view only portions of source code that are relevant to certain keywords. It simplifies code comprehension for programmers developing software for multiple architectures. RDT also offers a remote launching support that relies on Worqbench services for managing the compilation, execution, and debugging of grid applications on multiple remote machines. This remote support is transparent and seamless to the programmers; and it streamlines the process of application development with multiple grid resources.

3.6 Architecture and Design Summary

This chapter has presented the architecture and design of a new software infrastructure that supports the development of e-Science and grid applications. The infrastructure, ISENGARD, consists of three components, namely GridDebug, Worqbench, and Remote Development Tools (RDT). The chapter specifically discusses the design objectives of ISENGARD (Section 3.1) and its high-level architecture (Section 3.2). Then, it describes the design of GridDebug (Section 3.3), Worqbench (Section 3.4), and Remote Development Tools (Section 3.5) in detail.

ISENGARD has been designed according to the design objectives listed in Section 3.1 to meet the challenges that make developing grid software difficult. As discussed in Section 3.2, it employs a modular and layered software architecture with a collection of "pluggable" grid development tools as opposed to a single monolithic system. Specifically, the infrastructure consists of distinct software tools, modules, and plug-ins with APIs and bindings. ISENGARD components communicate to one another through standard and

well-defined interfaces. The components are loosely coupled and their explicit interfaces enable them to be used individually.

GridDebug (Section 3.3) is a generic debugging framework designed and built as an *extension* to grid middleware. The design simplifies the process of augmenting existing grid middleware with a debugging service. It provides the specification and implementation of a standard set of application programming interface (API) suitable for debugging and testing computational grid applications. The API is generic and it could support a number of tools and middleware. In addition, GridDebug provides an agent-based remote debugging with a callback mechanism to address non-trivial issues in debugging grid applications. For example, job scheduling and access restrictions across a hierarchy of resources.

Worqbench (Section 3.4) is an integrated and modular framework that links IDEs with grid middleware for e-Science application development. The Worqbench design is generic and it can accommodate and support other client applications besides IDEs, such as web portals or CLI programs. The generic design also allows the framework to support a variety of grid middleware. Worqbench maintains a system model that represents common grid application development scenarios. Worqbench provides various services that correspond to the stages in the implementation phase of a software development life cycle (Section 2.1). These services could be adopted as open standards of grid development services that can work with various middleware stacks.

Remote Development Tools (Section 3.5) are a set of modules and a user-level framework that augment an IDE for grid application development. RDT utilizes standard functions of the host IDE and provides features such as a multi-view code editor and a remote launching mechanism. It extends the IDE support for software development on distributed and heterogeneous grid resources. RDT provides a runtime programmable framework with a system model, API, and an event mechanism. Developers can write scripts that utilize RDT and IDE APIs to call or change the functionality of the IDE during its execution. It allows the automation of a variety of programming activities that become tedious if performed manually.

The next chapter presents a detailed description of the implementation of ISENGARD software components: GridDebug, Worqbench, and Remote Development Tools.

Chapter 4

System Implementation

This chapter presents the prototype implementation of ISENGARD which consists of three software components: GridDebug, Worqbench, and Remote Development Tools. We describe the implementation details of the GridDebug framework in Section 4.1, followed by the details of Worqbench in Section 4.2 and Remote Development Tools in Section 4.3. Finally, a summary section of the ISENGARD system implementation is given in Section 4.4.

4.1 GridDebug Implementation

The implementation of GridDebug provides an extension to the middleware in the form of a grid debugging service for finding, identifying, and fixing software bugs in grid applications. We have developed and implemented GridDebug using a number of popular software toolkits: WSRF (Section 2.5.2), Globus Toolkit 4 (Section 2.6.1), and GDB/GDBServer [GDB, 2007]. The main debug API library is implemented in Java, which provides an architecture neutral and portable programming language. We have also implemented a number of debug clients in various languages such as Python, Ruby, and Java.

Figure 4.1 gives an overview of GridDebug components and the implementation based on the design presented in Section 3.3. As described in

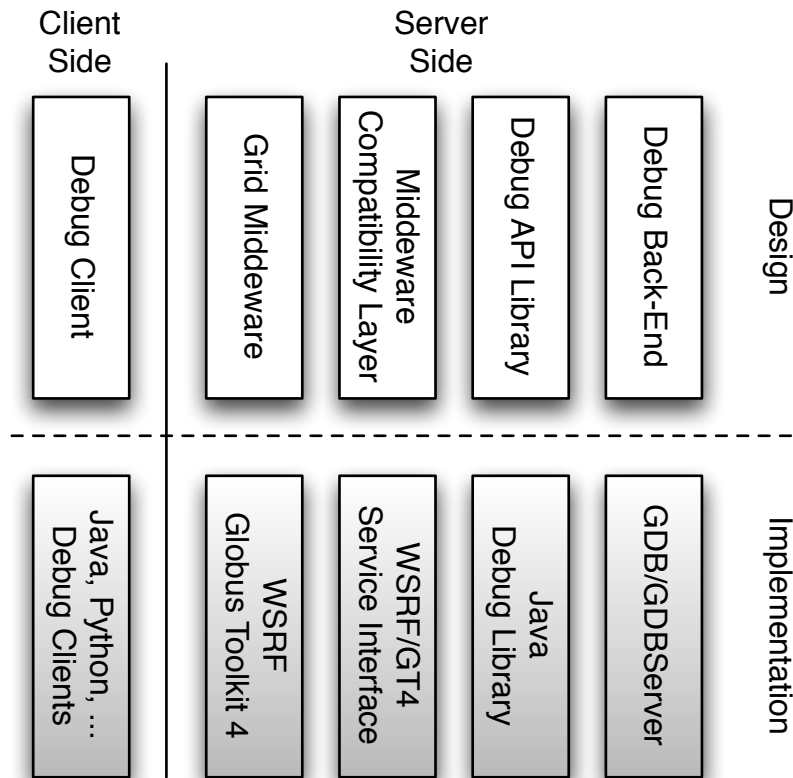


Figure 4.1: GridDebug implementation overview

that section, the design of GridDebug is generic and modular. Thus, it is possible to utilize other grid middleware and debug back-ends without any significant changes. Importantly, it simplifies extending existing middleware with a debugging service.

The implementation of GridDebug is described in greater detail in the following sections. Sections 4.1.1 and 4.1.3 describe the grid middleware service and the debug back-end respectively. The discussion on the GridDebug library and the list of API are given in Section 4.1.2.

4.1.1 Middleware Service

The GridDebug service is a WSRF web service that is hosted by the Globus Toolkit 4 WSRF container. The service wraps the middleware-independent debug library and exposes its functionality to WSRF-compatible or GT4

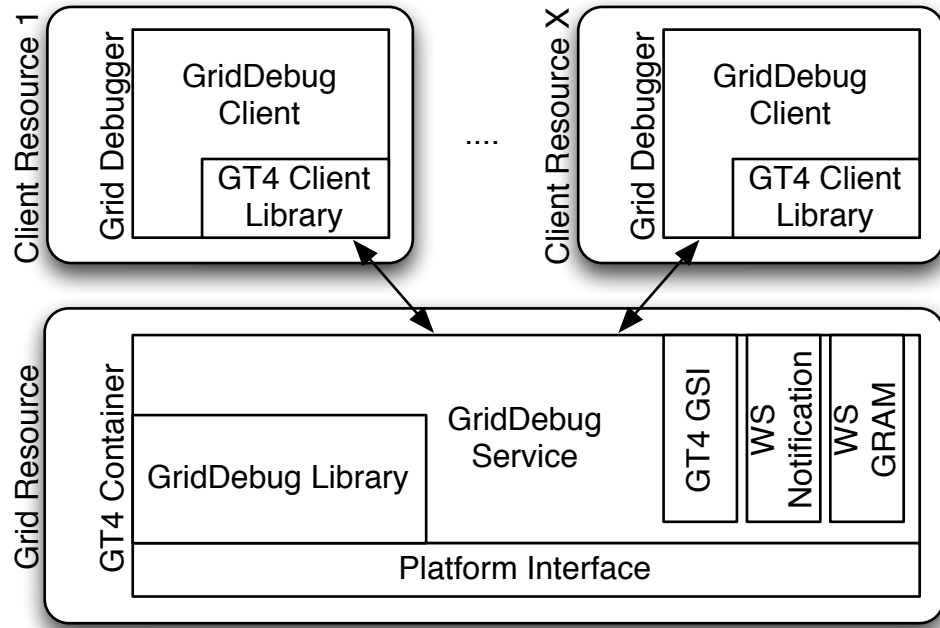


Figure 4.2: Interaction between GridDebug client and server components

client tools. The service translates GT4 specific API calls and objects to the generic GridDebug counterparts. GridDebug has been designed to leverage common and standard grid services, for example, security, notification, and execution services. Both client-side and server-side implementations utilize various GT4 services and development libraries such as WS-Notification, WS GRAM, and Grid Security Infrastructure (GSI).

The event notification including the application input/output mechanism is handled by WS-Notification whilst WS GRAM provides the necessary service to invoke the application and the back-end debugger. The security mechanism for message transport and authentication is provided by the GT4 GSI. By leveraging these services, existing GT4 client tools can utilize GridDebug functionality without extensive modifications. More importantly, GridDebug does not require changes to the grid security policy in place. Figure 4.2 illustrates the interaction between GridDebug client-side tools and server-side components which consist of various GT4 and GridDebug services.

The GridDebug debugging service has been implemented using a design

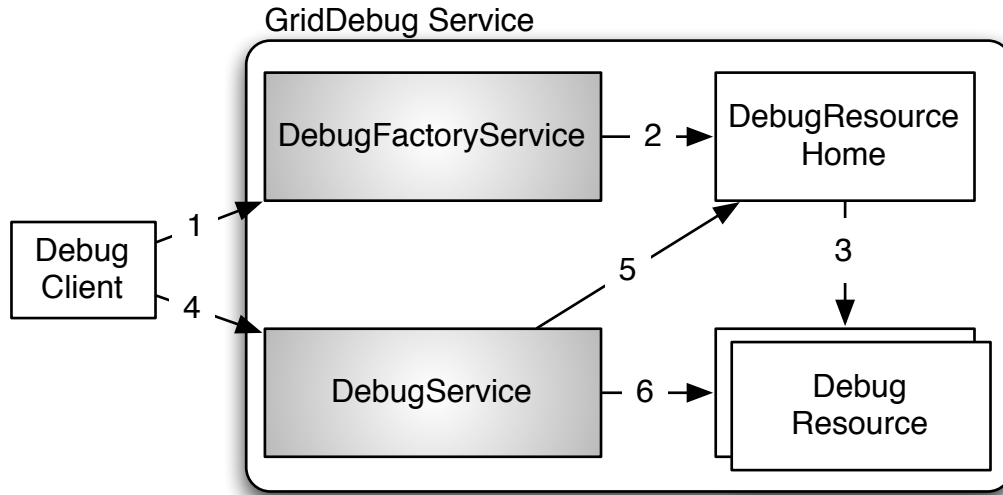


Figure 4.3: Components of GridDebug service implementation

pattern known as the factory/instance pattern [Sotomayor and Childers, 2005]. It allows the service to handle and manage multiple debugging sessions invoked by a number of developers. The design pattern implementation of the service consists of four major components, namely DebugResource, DebugResourceHome, DebugFactoryService, and DebugService. The relationship between the components is depicted in Figure 4.3 and the component descriptions are given as follows:

- The DebugResource is a WSRF resource that holds state information of a debugging session for an individual authorized user. It is the interface to the debug library and the underlying debug back-end. The DebugResource stores the actual DebugSession object (Section 3.3.2) and event information. Each DebugResource has a unique key to distinguish between different debugging sessions.
- The DebugResourceHome is a singleton entity that manages DebugResource instances. It maintains the key/object mapping information for DebugResource instances. The DebugResourceHome is used by the DebugFactoryService to create new resources. It is also utilized by the DebugService to find a resource with a given key.

- The `DebugFactoryService` is a web service that provides users with an interface to create and initialize new `DebugResources`. It returns an endpoint reference (EPR) to a `DebugResource` with its unique key. A `GridDebug` client performs various debugging operations using the EPR and the `DebugService`.
- The `DebugService` is a web service that provides an access point for invoking debugging operations on a `DebugResource`. It maps web service invocations to `GridDebug` API on a particular `DebugResource`.

The interaction between the client and the `GridDebug` service can be outlined as follows (Figure 4.3):

1. The client initiates a debugging session, connects to the `DebugFactoryService`, and requests an instance of `DebugResource`.
2. The `DebugFactoryService` utilizes the `DebugResourceHome` to create a new resource and returns an EPR to the client.
3. The `DebugResourceHome` creates a new `DebugResource` and updates its mapping information.
4. The client connects to the `DebugService` and requests debug operations to be invoked on a particular resource.
5. The `DebugService` uses the `DebugResourceHome` to find the resource.
6. The `DebugService` calls the `GridDebug` API on the resource.

The implementation of the `DebugResource` is linked with the `GridDebug` library which provides grid debugging functionality. The library is not GT4 specific and it has been designed to be independent of the grid middleware. Section 4.1.2 describes the `GridDebug` library and the debug API in greater detail.

4.1.2 GridDebug Library and API

The core debugging functionality of GridDebug is encapsulated in a middleware-independent debug library. It provides API for common debugging operations such as creating/deleting breakpoints, inspecting variables, and controlling program executions. The library is implemented in Java and it consists of five Java classes that are described as follows (Figure 4.4):

- **DebugSession.** It is the main class that serves as an interface to the grid middleware layer. The `DebugResource` accesses the debug back-end through this class. The `DebugSession` is an aggregator class and it holds instance variables to the other four Java classes: `IDebugger`, `DebugConfig`, `DebugEventManager`, and `RemoteDebugManager`.
- **IDebugger.** It is the class that provides the GridDebug API specification and serves as an interface to the debug back-end. The class defines a set of debug APIs that needs to be implemented by the underlying debugger. The API describes methods and data structures using an object-oriented paradigm. Various classes are defined to represent entities such as processes, breakpoints, and stack frames.
- **DebugConfig.** The class defines several configuration variables that allow a user to customize the initialization parameters and behaviour of the debug back-end. It provides methods to store and query the variables. The actual configuration variables need to be specified by the debug back-end implementation. It can specify necessary or optional configuration parameters such as home directory information, a path to an auxiliary program, TCP/IP port numbers, and so forth.
- **DebugEventManager.** The class is responsible for managing debug events such as breakpoint hit and error events. In addition, it also manages output messages which are treated as events. This event-based mechanism leads to an abstract and minimal coupling between interdependent components [Riehle, 1996]. The class consists of an event queue and a set of methods to post and retrieve debug events.

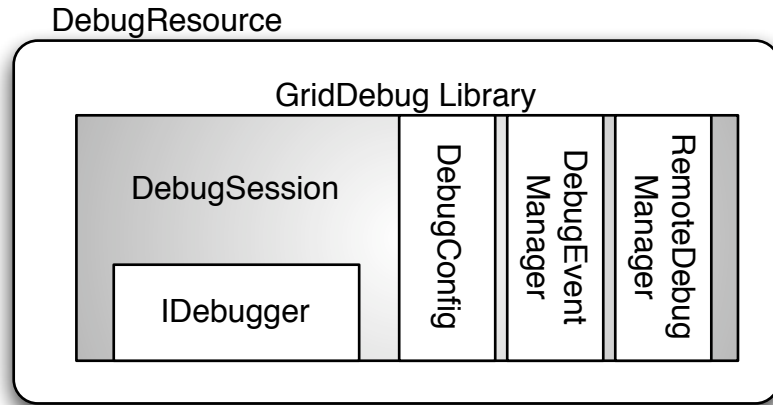


Figure 4.4: GridDebug library components

The debug back-end posts debug output and events into the queue. The `DebugSession` then forwards the events from the queue to WS-Notification subscribers.

- **RemoteDebugManager.** It is the class that is responsible for the remote debugging feature as described in Section 3.3.3. It manages the connection and forwards input/output streams between the debug back-end and the debug agent. This interaction is discussed in detail in Section 4.1.3.

The `IDebugger` specifies a set of GridDebug APIs based on the High Performance Debugging Forum (HPDF) standard [HPDF, 1998, Francioni and Pancake, 2000]. The HPDF standard is the result of significant research on an appropriate command set for a command-line interface (CLI) debugger, and it serves as a sound base for a debug library interface. HPDF commands, for example, `load`, `defset`, and `step` are implemented as API methods. Commands such as `help`, `alias`, and `history` are only peculiar to CLI tools and they are not translated into methods. A complete list of HPDF debug commands and the discussion on their syntax and semantics are given in HPDF [1998]. The abstract methods as specified by the `IDebugger` must be actualized by the debug back-end. It needs to extend the `IDebugger` class and provides a concrete implementation.

The GridDebug API methods are named according to their HPDF command counterparts. The methods follow the specification as set by the HPDF standard and they can be classified into six categories. Specifically, the categories are: debugger initialization and termination (Section 4.1.2.1); process sets (Section 4.1.2.2); actionpoints (Section 4.1.2.3); execution control (Section 4.1.2.4); program information (Section 4.1.2.5); and data display and manipulation (Section 4.1.2.6). The complete GridDebug API methods are given in Appendix A.

4.1.2.1 Debugger Initialization and Termination

Table A.1 lists methods which are associated with debugger initialization and termination. If the debug back-end supports multi-process debugging, a user can specify the number of processes that should be set up to run the program executable by calling the appropriate `load()` method. The `run()` method accepts an optional array of program arguments. The debugging session is terminated by calling the `exit()` method.

4.1.2.2 Process Sets

GridDebug specifies methods to support multi-process debugging (Table A.2), provided that the functionality is implemented by the debug back-end. A number of debugged processes can be grouped into a process set by calling the `defSet()` method with a set name and an array of process identifiers. A special set called `all` is defined to contain all processes. When a debugging session is started, the default target set is the `all` set, however, a user can change it by invoking the `focus()` method. Process sets can be deleted, without terminating the processes, by calling the `undefSet()` and `undefSetAll()` methods. The `viewSet()` method returns a group of set members as an array of `DebugProcesses`. The `DebugProcess` class encapsulates a debugged process.

4.1.2.3 Actionpoints

Actionpoint methods are used to specify the location or condition at which the flow of program execution should be suspended by the debugger. GridDebug defines two variants of actionpoints: breakpoints and watchpoints. A breakpoint hit event is triggered when a program reaches a particular location and a watchpoint hit event is triggered whenever the value of a variable changes. A `DebugActionpoint` class encapsulates an actionpoint including its specification, identifier, and type. The `actions()` methods can be invoked to get a list of actionpoints. GridDebug also supplies methods to delete, disable, or enable actionpoints. An actionpoint can be specified to affect all processes in a particular named process set. If a set is not specified, the actionpoint is defined for all processes in the currently focused process set. Table A.4 lists the methods associated with actionpoints.

4.1.2.4 Execution Control

GridDebug specifies a number of methods (Table A.5) that correspond to common execution control operations such as step, step over, step finish, halt, and continue. The `step()` and `stepOver()` methods accept an optional count argument that indicates the number of times that the operation needs to be performed. A user can also control the execution of all processes in a particular named process set by invoking the appropriate methods. If a set is not specified, the operation is performed on the currently focused process set.

4.1.2.5 Program Information

GridDebug specifies a number of API methods that are associated with program information. They allow a user to inspect the current listing of call stacks (`where()`) and navigate through the stack frames (`up()` and `down()`). A `DebugStackFrame` class encapsulates a call stack including its corresponding information. The methods associated with program information are listed in Table A.6.



Figure 4.5: Class diagram for GDBDebugger

4.1.2.6 Data Display and Manipulation

Table A.3 lists data display and manipulation methods. A user can print an expression (`print()`) and change the value of a program variable (`assign()`). A list of program variables that are valid and visible when the application is suspended can also be retrieved by calling the `listVariables()` method. It returns an array of read-only `DebugVariables`. The `DebugVariable` class has been defined to store a key/value pair of a variable.

4.1.3 GridDebug Back-End

The debug back-end is the software component that performs the actual debugging operations as defined by the GridDebug API. It extends the `IDebugger` class and provides an implementation of the API methods. We have developed a back-end debugger for GridDebug by utilizing GDB and GDBServer [GDB, 2007] with a small alteration. The necessary modification improves the debugger’s capability to handle situations where a job scheduler is employed to execute applications.

The GridDebug back-end is implemented by the `GDBDebugger` class. It extends the `AbstractDebugger` class, which in turn implements the `IDebugger` interface class. The class diagram is shown in Figure 4.5. The `AbstractDebugger` class contains common internal data structures and generic methods that can be used by concrete subclasses. The `GDBDebugger` class is the class that translates GridDebug API, as specified by `IDebugger`, into GDB commands. It utilizes the GDB/MI machine interface to communicate with GDB.

The GNU Debugger (GDB) is the de facto source-level debugger used on many computer architectures [GDB, 2007]. It can debug applications written in programming languages such as C, C++, Java, and Fortran. GDB offers

several advantages that make it suitable for a GridDebug back-end debugger. Specifically, they are:

- A machine interface called GDB/MI. It is a machine oriented text interface to GDB and it enables GDB to act as a software-controllable back-end debugger. The GDB/MI provides means for a high-level debugger to be built on top of GDB.
- A remote debugging feature, which is handled by an auxiliary program called GDBServer. This feature allows a programmer running GDB on one machine to debug an application on another remote machine. The GDBServer program running on the remote machine acts as a debug agent that performs process trace (ptrace) operations on the debugged application.
- Debugging support for a large number of computer architectures including Alpha, x86, x86-64, Itanium/IA-64, Motorola 68000, MIPS, SPARC, and PowerPC. This feature is advantageous in a grid environment composed of heterogeneous resources with different hardware architectures.

Figure 4.6 depicts the interaction between GDB and GDBServer; and how they operate. The interaction is outlined as follows:

1. A user logs in to a remote machine and invokes GDBServer with a program to be debugged. GDBServer then waits for a connection from GDB.
2. GDB is started on a programmer's desktop.
3. The debugging symbol file of the program is loaded in GDB.
4. The user instructs GDB to connect to the remote target using a TCP/IP connection. The user specifies the host name and the port number.
5. The user debugs the remote program as usual with GDB as if the debugged program runs on the programmer's desktop.

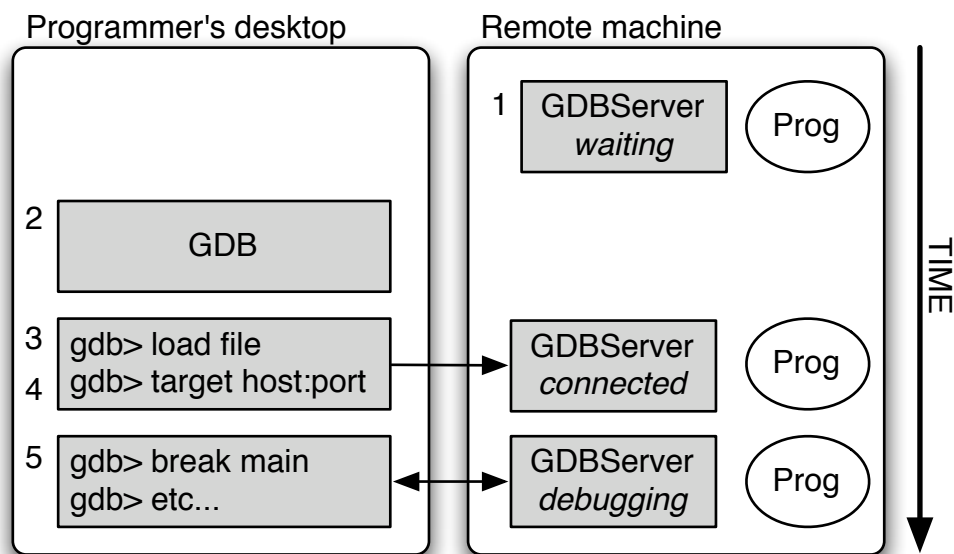


Figure 4.6: Interaction between GDB and GDBServer

Remote debugging using GDBServer has a number of advantages:

- GDBServer only acts as an agent that relays ptrace messages between GDB and the application. The primary debugging logic is still performed by GDB. As a result, the size of GDBServer is very small and it does not overload remote nodes.
- The debugging symbol resolution is performed at the GDB end thus a program on a remote machine does not need to be compiled with symbols included. However, the user must have another copy of the program with the debugging symbols on the local machine.
- GDB and GDBServer support debugging of cross-compiled applications. This allows the user to debug a program on a remote machine that has a different computer architecture from the user's desktop.

Nevertheless, in the grid environment, GDBServer is not suitable for remote debugging since it does not address issues such as security, job scheduling, and hierarchical resources. To solve this problem, we have extended GDBServer with a callback mechanism. This extension allows GDBServer

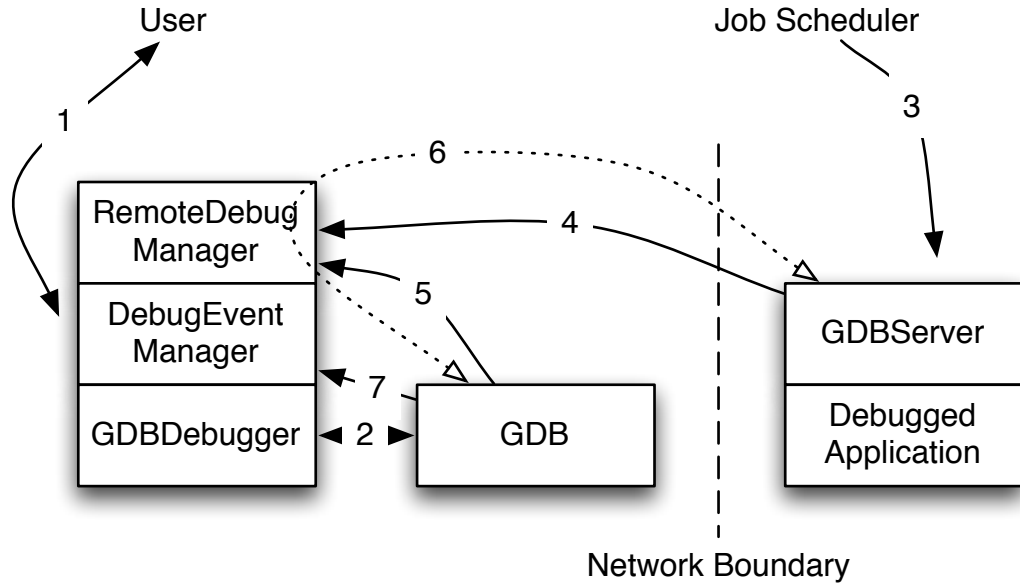


Figure 4.7: GridDebug debugging session

to initiate the remote debugging connection when it is ready to debug an application. Thus, this technique alleviates the need to constantly poll the job scheduler to check whether an application has been started. More importantly, it also adds another layer of security since the debugging activity is initiated by the application rather than by the programmer, thus a user cannot debug another user's processes.

GridDebug employs a remote debug manager to organize the communication between the back-end debugger and the debug agent. In the current implementation, the manager that is realized by the `RemoteDebugManager` class, opens two TCP/IP ports to accept incoming connections from GDB and GDBServer. Figure 4.7 shows a GridDebug debugging session example that details the interaction between the manager, GDB, and GDBServer. The figure is described as follows:

1. A user connects to the GridDebug service and initiates a session.
2. The service starts a GDB debugging session. The `GDBDebugger` serves as an interface to GDB.
3. At a scheduled time, the job scheduler executes a job that consists of

GDBServer and an application. The connection information is also supplied to GDBServer.

4. GDBServer runs and immediately suspends the application. It then connects to the `RemoteDebugManager`.
5. After GDBServer is connected, the `GDBDebugger` instructs GDB to connect to the `RemoteDebugManager`.
6. `RemoteDebugManager` acts as a transparent proxy by linking the two connections. GDB communicates with GDBServer.
7. The `DebugEventManager` receives debug events and output messages from GDB and forwards them to the user.

To demonstrate the use of GridDebug and its API, a number of debug clients have also been implemented in various languages such as Java and Python. These clients are described in Chapter 5.

4.2 Worqbench Implementation

As discussed in Section 3.4, Worqbench provides a framework that offers an API and high-level services that bridge between IDEs and current grid middleware, with a focus on e-Science application development. Worqbench has been designed to be modular, and is independent of the middleware and the IDEs. It allows developers to utilize a set of grid resources where each resource is managed by different grid middleware.

Worqbench is written in Ruby [Ruby, 2008] and is built on top of Rails. Rails is a full-stack framework for developing database-backed web applications according to the Model-View-Controller (MVC) pattern [Rails, 2008]. Worqbench can be implemented in other languages such as Java with its web-related frameworks (servlets and portlets). However, Rails is selected for the implementation because it is a lightweight framework compared with the Java counterparts [Rustad, 2005]. In addition, Rails offers several features that serve as a sound starting point [Tate, 2006, Fernandez, 2007], such as:

- It is an integrated platform for building web portals with rich user interfaces. Rails provides a web template engine, a client-side scripting support, various HTML helper methods, and an Ajax user interface library [Garrett, 2005].
- It has built-in support for web service protocols such as XML-RPC [Winer, 1999] and SOAP [W3C, 2007a]. Additionally, it also supports Representational State Transfer (REST), which is a software architecture style for distributed systems such as the World Wide Web [Fielding, 2000].
- It contains a database persistence layer called ActiveRecord [Marshall et al., 2007] that implements the object-relational mapping (ORM) pattern. In this pattern, a row in a database table or view is wrapped by an object, which encapsulates the database access and adds domain logic on that data [Fowler, 2002]. Rails has various adapters for database back-ends such as PostgreSQL [PostgreSQL, 2008], MySQL [MySQL, 2008], and SQLite [SQLite, 2008].
- A plug-in system that lets users and developers add new functionality into the Rails framework. Rails can also use various third party Ruby libraries. Furthermore, Java and Microsoft .NET libraries can also be utilized by running Rails under Ruby interpreters such as JRuby [JRuby, 2008] and IronRuby [IronRuby, 2008].

In addition, we also use the Ruby Source Control Management library [RSCM, 2006] that provides common high-level API methods for accessing different source code repositories such as CVS [CVS, 2006] and Subversion [Subversion, 2007].

The Worqbench implementation is discussed in greater detail in the following sections. Section 4.2.1 describes the three possible Worqbench client/server arrangements. Sections 4.2.2 and 4.2.3 describe the Worqbench client connectors and middleware adapters respectively. The Worqbench entities and their related API methods are discussed in Section 4.2.4 whilst

Section 4.2.5 details the Worqbench services for grid application development.

4.2.1 Client/Server Arrangements

Worqbench links an IDE to grid middleware in a three-tier model as shown in Figure 3.5 in Section 3.4. This three-tier model ensures flexibility in running Worqbench with different client/server arrangements. Figure 4.8 shows three different arrangements that are described as follows:

1. Worqbench is run on a separate machine from client and server computers. In this arrangement, a single Worqbench installation serves requests from multiple clients to multiple grid resources. It can simplify user and resource management since there is only one Worqbench server to administer.
2. Worqbench is executed on the same machine as the client. In this set up, each client has its own Worqbench server process that manages the connection to grid resources. It is advantageous to users since it eliminates potential problems such as client-side firewall issues.
3. Worqbench is run on the same machine as the server. This arrangement configures a Worqbench server process to accept remote connection for each grid resource. This set up is useful if the resource is put in a private and restricted environment such as a personal grid test bed.

These arrangements allow Worqbench to meet different network topology requirements and restrictions imposed by various grid sites and users. In addition, Worqbench utilizes storage servers for its database and code repository. As a result, it is possible to minimize the load of the Worqbench server by employing external database and code repository servers.

4.2.2 Client Connectors

Worqbench is a web application that relies on a web server to accept incoming client connection. It can use the Apache HTTP Server [Apache HTTPD,

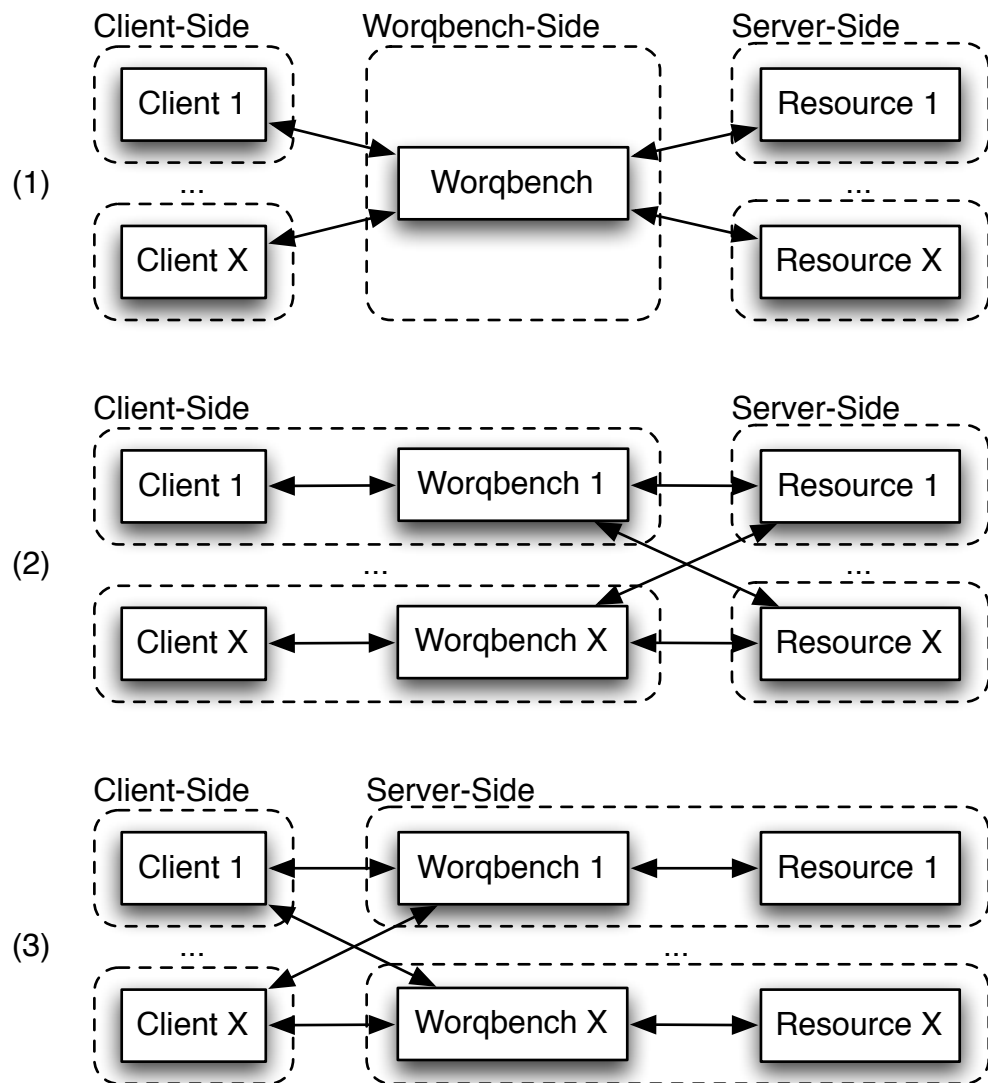


Figure 4.8: Worqbench client/server arrangements

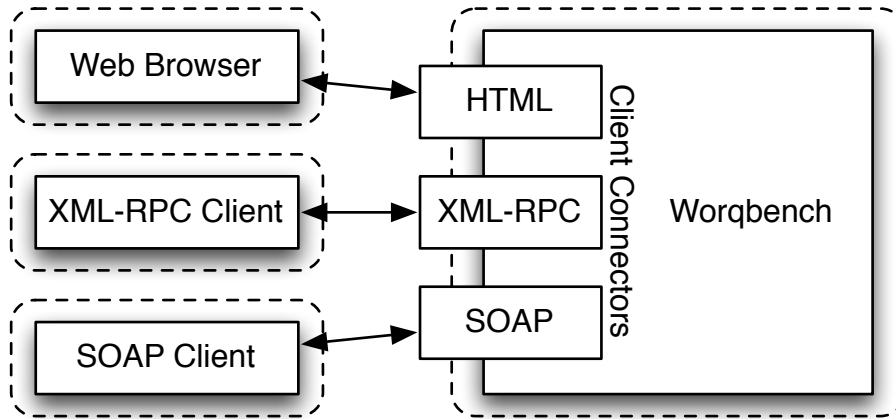


Figure 4.9: Client connectors

2008] for a wide scale deployment or WEBrick [WEBrick, 2008], a standard Ruby HTTP library, for personal use. We have implemented three client connectors for HTML pages, XML-RPC, and SOAP protocols (Figure 4.9). The HTML connector implements a web portal interface to Worqbench. This interface is beneficial since it allows users to access Worqbench functionality using a web browser without the need to install specialized software.

The XML-RPC and SOAP connectors provide a web service interface to Worqbench services. It allows web service clients to access and invoke Worqbench remote procedure calls. The XML-RPC and SOAP protocols support scalar data types such as booleans, integers, and strings; and vector data types such as arrays and hashes. In addition, the protocols utilize Base64 data encoding [Josefsson, 2006] for transmitting binary data such as application project files.

Worqbench employs a username-password pair for authentication and a session key for authorization. The key must be passed as an argument for every remote procedure call to Worqbench services. To illustrate this, Listing 4.1 shows a web service client script written in Python that uploads a project zip file to a Worqbench server. The script sets up an XML-RPC connection to the server (line 2) and obtains a session key by calling the `admin.GetKey()` method with a username-password pair (line 3). The script then reads the zip file into a variable (line 4) and uploads the data to the server by calling the


```
1 import xmlrpclib
2 server = xmlrpclib.Server('http://example.com:3000/service/↵
    api')
3 key = server.admin.GetKey('donny', 'password')
4 data = xmlrpclib.Binary(open('/tmp/HelloWorldCode.zip', 'r')↵
    .read())
5 server.project.UploadData(key, 'HelloWorld', data)
```

Listing 4.1: A Python script to upload a project zip file

`project.UploadData()` method (line 5). The method requires a valid session key, a name to an existing project entity (`HelloWorld`) that is associated with the zip file, and the data itself.

The network connection can be encrypted by running Worqbench with a web server that supports HTTPS with Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocol [Dierks and Rescorla, 2006]. The Worqbench client must also support the corresponding security protocol.

4.2.3 Grid Middleware Adapters

As of this writing, we have implemented adapters that allow Worqbench users to execute applications on SSH, GT4, and UNICORE resources (Figure 4.10). The GT4 adapter also supports application debugging by utilizing the GridDebug service. The adapters use libraries and command-line tools such as `globusrun-ws` and `uuc`, to communicate with the back-end middleware. As an illustration, Figure 4.11 shows an interaction between the GT4 adapter, Globus CLI tools, and GT4 services. The GT4 adapter utilizes command-line tools, for example, `globusrun-ws` for command execution, `globus-url-copy` for file transfer, and `gt4debugclient` for application debugging.

When an operation, for example file transfer or program execution, is invoked by a user, Worqbench spawns a non blocking execution thread and creates a data structure entity (`UserTask`) to encapsulate and represent the operation. The user can query the `UserTask` object for the status and output of the operation. In addition, the user can also send program input and debugging commands to the entity if it represents an execution or debugging operation. The `UserTask` entity and other Worqbench data structures are

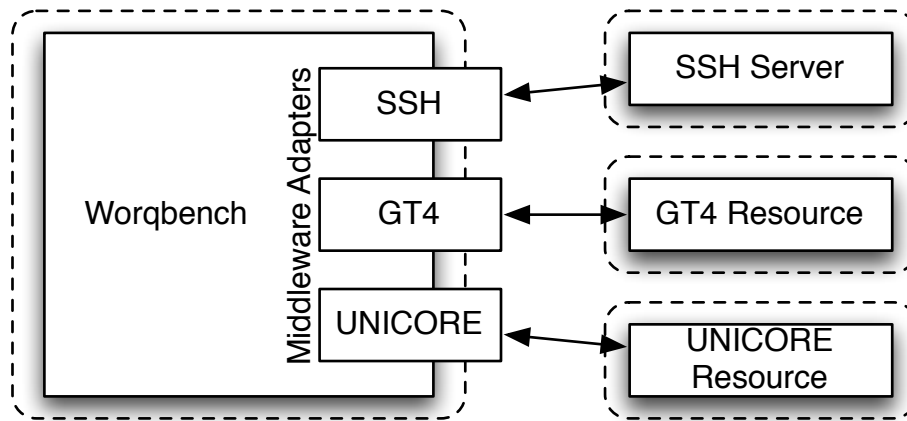


Figure 4.10: Middleware adapters

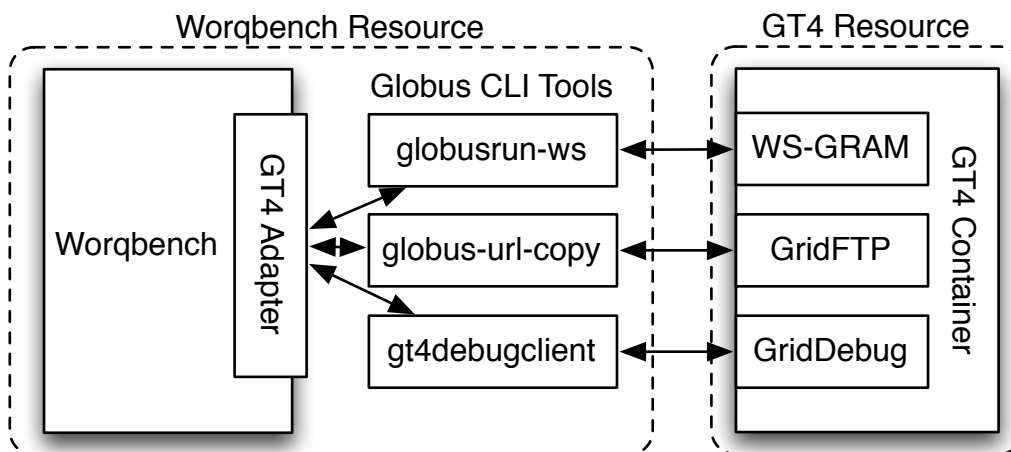


Figure 4.11: Interaction between Worqbench, CLI tools, and GT4 services

described in detail in Section 4.2.4.

4.2.4 Worqbench Data Structures and APIs

Worqbench defines several Rails model classes to implement the system entities (Section 3.4.2). Figure 4.12 shows the model classes and their public data members. The figure is an implementation-oriented diagram of Figure 3.7 in Section 3.4.2. The diagram uses Barker's Notation [Barker, 1990] to depict the cardinality of the class relationships.

The Worqbench entities can be accessed through a web portal and a set of web service APIs. By using a web browser, a user can perform operations such as creating new resource sets; uploading a project zip file and security certificates; and executing the application project on grid resources. The web service APIs enable the entities to be utilized, administered, and queried programmatically by invoking various remote procedure calls.

The class data members shown in Figure 4.12 represent standard attributes for the entities. Worqbench also implements a mechanism to store other metadata information or user-defined custom attributes. The classes have internal hashes that store these attributes in the form of key/value pairs. API methods, namely `SetAttr()` and `GetAttr()`, can be used to assign and retrieve the custom attributes.

The API methods of the Worqbench entity classes are listed in Appendix B whilst the discussion is given as follows:

- **Resource and User.** As discussed in Section 3.4.2, there are two resource entities in Worqbench: Resource and UserResource. The Resource class encapsulates resource specific information such as host address and port number. Other system resource information, for example compiler and library paths, can be specified in the `configuration` attribute. The UserResource class extends the Resource class for a particular user with user specific data, for example, login name and security certificate. Worqbench also defines two types of users: administrator and regular users. Administrators are specified when Worqbench is installed and

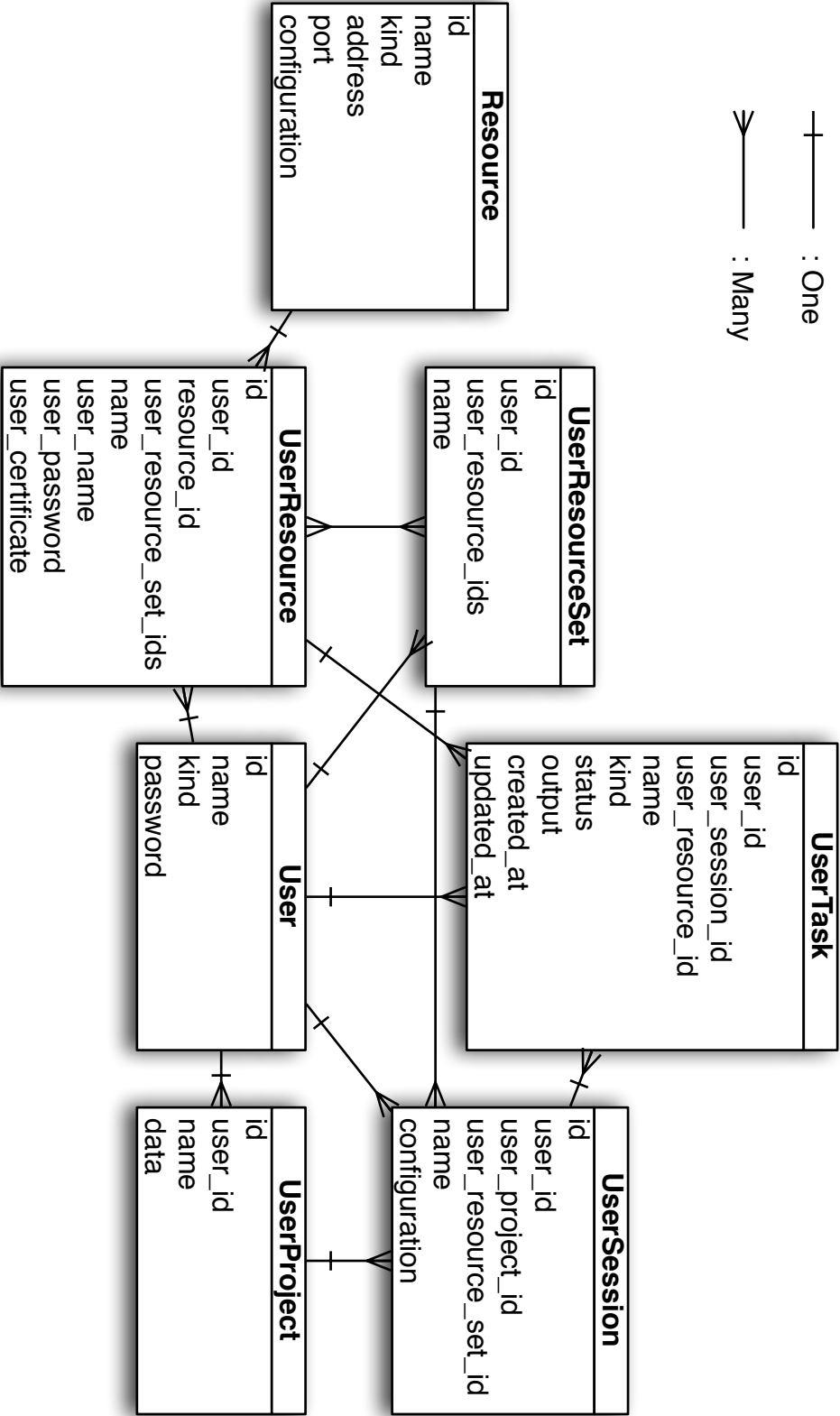


Figure 4.12: Worqbench model classes

they can create new regular users (the User entities) and system resources (the Resource entities). API methods and standard attributes for the Resource and User entities are listed in Table B.1.

- **UserResource.** The UserResource class represents one machine or resource that can be accessed by a user. It keeps login information for the user in its `user_name`, `user_password`, and `user_certificate` attributes. User specific attributes such as home directory and path information can be specified by invoking the `resource.SetAttr()` method. UserResource API methods and standard attributes are listed in Table B.2.
- **UserProject.** The UserProject class represents an application project of a user. It stores a zip file containing all necessary files for a development project in its `data` attribute. Custom attributes such as project version information can be specified by the user by calling the `project.SetAttr()` method. Table B.3 lists UserProject API methods and data members.
- **UserResourceSet.** A collection of grid resources is represented by the UserResourceSet class. It maintains an internal array that stores membership references to UserResource entities (`user_resource_ids`). A user can add or remove resources from a set by calling the `set.AddResource()` and `set.RemoveResource()` methods respectively. A list of UserResourceSet attributes and methods is given in Table B.4.
- **UserSession.** The UserSession class represents a development session where a user develops an application to be executed on a resource set. The `configuration` attribute stores information such as compilation and execution commands for a session. UserSession data members and API methods are listed in Table B.5.
- **UserTask.** A remote operation is represented by the UserTask class. A UserTask object is created automatically by Worqbench when a user invokes an operation such as compiling or executing an application project. Table B.6 lists all UserTask standard attributes and methods.

```
1 BUILD_START
2 export PATH=/usr/local/bin:/usr/bin:/bin
3 (cd directory && make clean all)
4 BUILD_END
5
6 RUN_START
7 (cd binary && ./application)
8 RUN_END
9
10 CUSTOM_COMMAND_START
11 OS=`uname `
12 case $OS in
13   "Linux") echo 'OS is Linux';;
14   "Darwin") echo 'OS is Mac OS X';;
15   *) echo 'Unknown';;
16 esac
17 CUSTOM_COMMAND_END
```

Listing 4.2: An example of a UserSession's configuration

All Worqbench API methods, with the exception of `admin.GetKey()`, require a session key as an argument. It acts as a security authorization token and it is different between users. The key is obtained by calling the `admin.GetKey()` method with the user's Worqbench login information.

Listing 4.2 shows an example of a UserSession's configuration with three user-defined operations. In the current implementation, an operation is specified as a shell script that is passed to a command-line interpreter on the target resources. Lines 2-3 are executed when the method `packaging.Build()` (a part of the Worqbench Packaging Service API) is called. Similarly, lines 7 and 11-16 are executed when the methods `launching.Run()` and `launching.-Command('CUSTOM_COMMAND')` are invoked respectively. Worqbench services are described in the next section (Section 4.2.5).

4.2.5 Worqbench Services

Worqbench implements grid development functions as web services that are invoked through remote procedure calls. The API methods require the security authorization key and a UserSession object. The methods operate on a UserSession object with its application project and resource set. As described

in Section 3.4.1.5, Worqbench provides three sets of APIs that correspond to the engineering stages in a software development life cycle: Packaging Service API (Section 4.2.5.1), Launching Service API (Section 4.2.5.2), and Editing Service API (Section 4.2.5.3).

4.2.5.1 Packaging Service API

The Packaging Service provides methods for transferring and compiling an application project on a set of remote resources. The API methods are described as follows:

packaging.Transfer() copies a project zip file to user's home directories on remote resources. The default operation can be altered by editing the **configuration** attribute in the **UserSession**'s object.

packaging.Init() initializes a project zip file. Typically, it decompresses the zip file into a project folder. Additional initialization steps can be specified by the user in the **UserSession**'s **configuration**.

packaging.Build() compiles an application in a project folder according to the compilation commands as specified in the **UserSession**'s **configuration** attribute. The target resources need to have the required development tools and libraries to build the application.

packaging.Clean() deletes the project zip file and folder on remote resources. The user can instruct additional clean up operations to be performed by changing the **UserSession**'s **configuration** attribute.

4.2.5.2 Launching Service API

The Launching Service provides API methods that allow a Worqbench user to execute or debug (utilizing the GridDebug service) an application on multiple resources. The method descriptions are given as follows:

launching.Run() runs the compiled application software on remote resources. It generates a **UserTask** object for each resource. The method

performs the execution commands as specified in the `UserSession`'s `configuration`. The program arguments are also specified in the `configuration` attribute.

launching.Debug() executes the compiled program on remote resources in debug mode. The user needs to specify the debugging commands in the `UserSession`'s `configuration`. The operation creates a `UserTask` object to represent the execution on each resource.

launching.Command() executes an arbitrary shell command script as specified by the user in the `UserSession`'s `configuration` on remote resources. The script execution on each resource is represented by a `UserTask` object.

4.2.5.3 Editing Service API

The Editing Service offers a set of API methods for managing a software project including its multiple revisions on a user's local computer. The service depends on the `UserProject`'s `data` attribute that must be populated by calling the `project.UploadData()` method. The Editing Service methods are described as follows:

editing.Checkout() checks out or updates the project data from the Worqbench code repository to a local working copy. The user needs to specify the name of a local directory and an optional revision number.

editing.Commit() checks in or commits the modified source code files in a local working directory to the Worqbench code repository. The method accepts an optional commit message as its argument.

editing.GetRevisions() lists all revisions of a `UserProject`'s `data` including their timestamps and commit messages in the code repository.

editing.IsUpdated() checks whether a local working copy of the project is in sync with the one in the Worqbench code repository.

Worqbench users can utilize their editing tools to edit the source code files in a local working directory. They can also use the corresponding source code management (SCM) client-side tools to view the history, diff changes, and message log of files. After the modification, the files can be committed to the Worqbench code repository by calling the `editing.Commit()` method. Alternatively, the whole project folder can be compressed and uploaded by calling the `project.UploadData()` method.

4.3 RDT Implementation

We have developed an implementation of Remote Development Tools (RDT) in the form of Eclipse plug-ins (Section 2.3.6). As introduced in Section 3.5, RDT augments the IDE through its plug-in extension mechanism and leverages the GridDebug framework and Worqbench services for grid application development. RDT provides a multi-view code editor, an application launching support for multiple remote resources, and an IDE scripting facility to automate a variety of programming tasks. RDT has been designed to be IDE-independent. Thus, it is possible to incorporate RDT into another IDE, for example, NetBeans (Section 2.3.5).

Eclipse RDT is implemented as several plug-ins written in Java that can be loaded into the Eclipse IDE. For the automation facility, the Eclipse RDT features an embedded interpreter for the Groovy programming language [Groovy, 2008]. Developers utilize the RDT framework API and event mechanism by writing scripts and event callback methods in Groovy. It is a dynamic object-oriented scripting language for the Java platform. Groovy has features similar to Python, Perl, and Ruby such as dynamic typing; late binding; reflection; native syntax for lists, maps, and regular expressions; and closures. It has a simpler syntax than Java, support for various markup languages, and powerful processing primitives. In addition, it can invoke and integrate seamlessly with all existing Java objects and libraries.

The implementation of RDT is discussed in detail in the following sections. Sections 4.3.1 and 4.3.2 describe the integration of RDT into Eclipse and its user interface. The RDT data structures and the framework API are

| Package name (non-UI layer) | RDT component |
|---|---------------|
| <code>org.eclipse.rdt.core</code> | Core |
| <code>org.eclipse.rdt.core.models</code> | Core |
| <code>org.eclipse.rdt.core.commands</code> | Core |
| <code>org.eclipse.rdt.launch</code> | Launch |
| Package name (UI layer) | RDT component |
| <code>org.eclipse.rdt.ui</code> | View |
| <code>org.eclipse.rdt.ui.views</code> | View |
| <code>org.eclipse.rdt.ui.preferences</code> | View |
| <code>org.eclipse.rdt.ui.editors</code> | Editor |

Table 4.1: Java packages that implement Eclipse RDT

outlined in Sections 4.3.3 and 4.3.4. Finally, Section 4.3.5 presents the RDT multi-view code editor.

4.3.1 IDE Integration

Eclipse RDT is implemented according to the Eclipse plug-in guidelines [Clayberg and Rubel, 2006]. It is divided into two software layers, UI and non-UI; and it consists of several Java packages. Table 4.1 lists the packages and the RDT components that they implement while Figure 4.13 gives an illustration of Eclipse RDT and Worqbench.

Eclipse defines a number of models such as `IFile`, `IFolder`, and `IProject` that provide abstractions for local files, folders, and projects respectively [Gamma and Beck, 2003]. RDT augments this Eclipse definition by providing other models, specifically `IProject`, `IResource`, `ISet`, `ISession`, and `ITask`. The `IProject` model acts as a thin wrapper around the Eclipse `IProject` and implements the necessary data structure as required by RDT. In the implementation, we have created these RDT models as wrapper classes around the Worqbench entities. The wrapper classes provide abstractions for Worqbench `UserProject`, `UserResource`, `UserResourceSet`, `UserSession`, and `UserTask` entities.

In addition, RDT also provides wrapper classes for debugging-related entities such as breakpoints, stack frames, and variables. These classes function as a translation layer between GridDebug and Eclipse-specific debug models.

Eclipse has the concept of a launch configuration, which is defined as a

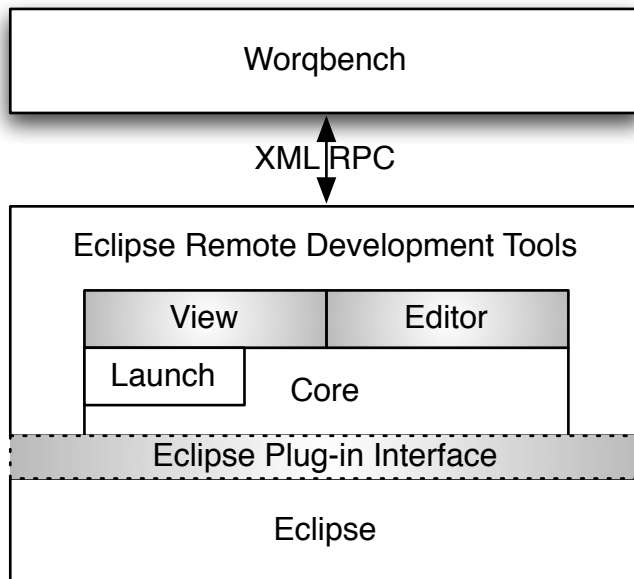


Figure 4.13: Eclipse RDT and Worqbench

description of how to run or debug a program [Szurszewski, 2003]. Eclipse has built-in launch configurations for launching a Java program, an applet, or another instance of Eclipse. RDT contributes a launch configuration plug-in that enables Eclipse to run and debug grid applications on a set of remote resources through Worqbench Launching Service. The plug-in also utilizes the Worqbench Packaging Service to transfer a project to the remote resources, initialize it, and build the application. In addition, the plug-in uses various launching and debugging support systems in Eclipse, such as an application output window, variable views, stack trace views, breakpoint markers, an editor with current line highlighting, and so forth.

RDT communicates with Worqbench using the XML-RPC client connector (Figure 4.13). RDT invokes synchronous XML-RPC calls to access and utilize Worqbench services and entities. The Worqbench connection can be encrypted by utilizing the HTTPS protocol.

4.3.2 User Interface

RDT has its own Eclipse perspective, the Remote Development perspective, that displays windows and editors related to grid application development. An Eclipse perspective is a collection of related windows and views for a specific task. RDT implements various Eclipse views that provide GUI management tools for remote resources, resource sets, projects, sessions, and tasks. These views are organized in the Remote Development perspective. One of the primary views is the Remote Resources view that allows programmers to create, edit, destroy, and manage remote resources and sets.

We have also implemented an Eclipse preferences dialog for configuring RDT settings and an Eclipse launch configuration dialog for launching remote grid applications. To run or debug a grid application, a programmer needs to specify the development session, the project, and the resource set on which the application will be built and executed. A multi-view code editor that extends a standard Eclipse editor has also been implemented. The editor is discussed in detail in Section 4.3.5.

4.3.3 RDT Data Structures

RDT defines and implements model classes as proxies for Worqbench entities. There are five RDT models, namely `IProject`, `IResource`, `ISet`, `ISession`, and `ITask`. They have the same data members as their corresponding Worqbench entities (Section 4.2.4) with additional data structures for RDT specific metadata. The additional data members are used to store RDT tagging information. API methods, namely `setTags()` and `getTags()`, can be used to assign and retrieve an object's tag. Figure 4.14 shows a class diagram for RDT models. There are three primary classes for each model: a class interface, a class implementation, and a singleton manager class.

The class interfaces specify method signatures that must be implemented by the model classes. The singleton managers provide administrative methods to create, delete, query, and access the entities. The entity and manager methods are discussed in Section 4.3.4.

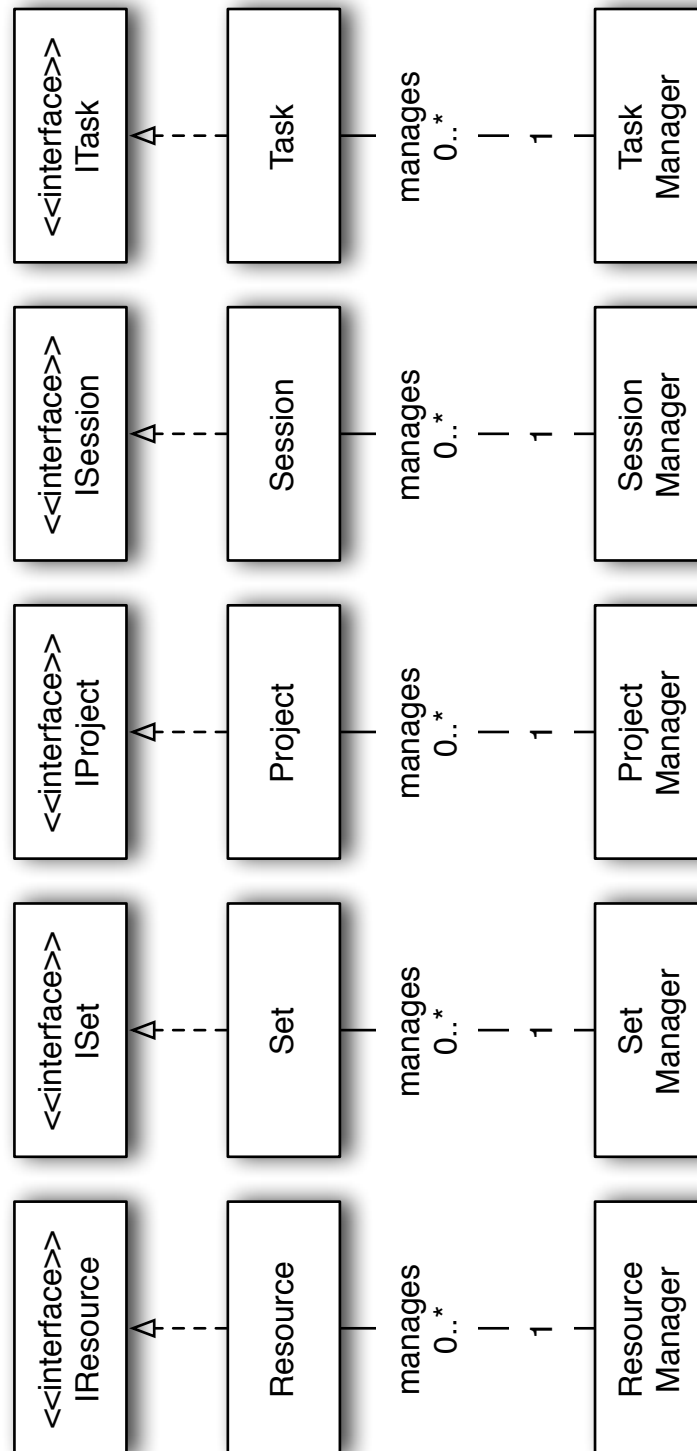


Figure 4.14: Class diagram for RDT models

4.3.4 RDT Framework API

The implementation of RDT includes an embedded interpreter for the Groovy programming language in the Eclipse IDE. It enables developers and users to program and customize the IDE during its runtime. The RDT application programming interface is implemented as Java objects and methods that can be called and accessed directly from within Groovy scripts. There are two ways to access the framework API:

- Through a Groovy Shell view. It is a Groovy command-line interpreter that is contained in a special Eclipse window view. It allows a programmer to evaluate Groovy expressions, invoke methods, define classes, and run RDT scripts that alter the behaviour and functionality of the IDE. The shell view is similar to the Python interpreter program or the Interactive Ruby Shell (IRB) [Ruby, 2008].
- Through an Eclipse event preferences window. It is an event configuration dialog where a developer can write and save RDT scripts for event callback methods. The scripts are standard Groovy scripts that can call and access methods in the framework API. Eclipse RDT will run the event scripts based on the occurrence of corresponding events. The Eclipse RDT event mechanism is described in the next section (Section 4.3.4.1).

Tables C.1, C.2, and C.3 in Appendix C list the currently implemented interfaces for the RDT entities and managers. In addition to RDT API, built-in Eclipse APIs including the graphical user interface widget (e.g. dialog boxes) APIs are also *callable* within the Groovy scripts. Listing 4.3 shows an example of utilizing the RDT API. The Groovy script prints the name of all grid resources located at Monash University. It also demonstrates dynamic data typing, regular expressions, and closures in Groovy.

The script calls the `ResourceManager.getResources()` method to store all RDT resources in an array called `allResources` (line 1). It defines a new array called `allMonashResources` (line 2). The script then iterates over the `allResources` array (lines 4-7). If a network address of a resource

```
1 allResources = ResourceManager.getResources();
2 allMonashResources = [];
3
4 for (resource in allResources) {
5     if (resource.getResourceAddr() ==~ /monash.edu.au/)
6         allMonashResources.add(resource.getResourceName());
7 }
8
9 allMonashResources.eachWithIndex() {obj, i -> println "${i}:↵
    ${obj}"};
```

Listing 4.3: A Groovy script that utilizes RDT API

contains the string `monash.edu.au` (line 5), the resource name is added to the `allMonashResources` array (line 6). At line 9, the script prints the members of `allMonashResources` array: the resource name and its corresponding array index.

4.3.4.1 Event Listener and Callback

RDT implements an event mechanism with user-defined callback methods. Internally, it has an event queue, an event listener, and a collection of objects that correlate with RDT events. A list of implemented RDT events is given in Table 4.2.

RDT adds an event object to the queue at the completion of related API methods. The event listener checks and removes the event from the queue and invokes the corresponding user-defined callback method. A user writes the callback methods in Groovy in the Eclipse RDT event preferences window. The callback methods have designated names such as `onFileChanged()`, `onProjectDestroyed()`, `onSetCreated()`, and so forth. In addition, there are special variables that are relevant and valid within the scope of the methods. For example, in the `onSetCreated()` method, a special variable called `name` contains the name of the newly created resource set. A number of examples of event callback methods are given in Chapter 5.

In the implementation, built-in Eclipse APIs, data structures, and GUI widgets are callable within the Groovy scripts. Thus, the callback methods can display a pop-up dialog box, close Eclipse perspectives, invoke the com-

| RDT events | Description |
|-------------------------------------|---|
| <code>ProjectCreatedEvent</code> | A new project has been created. |
| <code>ProjectDestroyedEvent</code> | A project has been deleted. |
| <code>ProjectChangedEvent</code> | A project's properties have been changed. |
| <code>ResourceCreatedEvent</code> | A new resource has been created. |
| <code>ResourceDestroyedEvent</code> | A resource has been deleted. |
| <code>ResourceAddedEvent</code> | A resource has been added to a set. |
| <code>ResourceRemovedEvent</code> | A resource has been removed from a set. |
| <code>SetCreatedEvent</code> | A new set has been created. |
| <code>SetDestroyedEvent</code> | A set has been deleted. |
| <code>FileCreatedEvent</code> | A new file has been created. |
| <code>FileDestroyedEvent</code> | A file has been deleted. |
| <code>FileChangedEvent</code> | A file's content has been changed. |
| <code>FolderCreatedEvent</code> | A new folder has been created. |
| <code>FolderDestroyedEvent</code> | A folder has been deleted. |
| <code>FolderChangedEvent</code> | A folder's content has been changed. |

Table 4.2: Eclipse RDT events

pile action, run an executable, and so forth based on the occurrence of RDT events.

4.3.5 RDT Multi-View Code Editor

As introduced in Section 3.5.2.3, the RDT multi-view editor has been designed to simplify code comprehension for programmers developing multi-platform software. It achieves this by showing only relevant portions of the code and hiding the irrelevant parts. This is in contrast to traditional text editors that normally display all contents from a file. We have implemented an Eclipse RDT multi-view editor by extending the standard Eclipse text editor.

The IDE provides a number of editors for text and source files with features such as code highlighting, search/replace, text folding, and so forth [Deva, 2005, Clayberg and Rubel, 2006]. In addition, it also provides the relevant Java classes, interfaces, and GUI controls (or widgets) for enhancing the editors.

The Eclipse RDT multi-view editor consists of a normal view and multiple sub-views (Figure 3.11 in Chapter 3). It works with a single instance of the file and multiple "in memory" data structures; and it does not keep a

separate physical copy of the file for each sub-view. A programmer can edit the source text in a view and the changes will be saved in the file. The problem of concurrency is avoided by allowing only one view to be active and editable at one time. The editor is implemented by creating a new GUI widget, utilizing regular expressions for pattern matching, and leveraging the text folding mechanism.

The new widget is written as a GUI window container that holds the code editor's multiple views. The editor partitions the file into several text sections according to a user-defined regular expression; and they are annotated with tags. The programmer requests a new sub-view by providing the editor with a list of tags. The editor then instantiates the sub-view that shows text sections that match the tags and hides non-matching sections. The text folding mechanism of the Eclipse editor is employed to show or hide the text sections in the sub-view [Deva, 2005]. The folding operations are performed *automatically* by the editor. This is different from the mechanism in traditional text editors, which requires manual user interaction to show or hide blocks of text.

A demonstration of the multi-view code editor is given in Chapter 5.

4.4 System Implementation Summary

This chapter has presented the prototype implementations of GridDebug, Worqbench, and Remote Development Tools that form a software infrastructure for e-Science and grid application development.

A prototype of GridDebug (Section 4.1) has been implemented by leveraging a number of popular software toolkits such as WSRF, Globus Toolkit 4, and GDB/GDBServer. The implementation provides a standard set of API suitable for debugging grid applications running on GT4 resources. It is secured by utilizing the GT4 GSI mechanism and it does not require changes to the grid security policy in place. The API can be used to build sophisticated or customized high-level grid debugging tools. The prototype utilizes the well-proven GDB/GDBServer debugger that allows it to support a large number of computer architectures and operating systems. In addition, with a

small extension to GDBServer, the debugging service can address non-trivial grid debugging issues such as job scheduling and access restrictions across a hierarchy of resources.

The implementation of Worqbench has been described in Section 4.2. It is flexible in accommodating different network topology requirements and it is extensible to support different types of clients and grid middleware. The Worqbench functionality is easily accessible by using a standard web browser and the current prototype supports protocols such as XML-RPC, SOAP, and REST. Other communication protocols can be supported by implementing appropriate connectors. Worqbench provides adapters for SSH, GT4, and UNICORE resources. More importantly, its versatile design allows the prototype to support future grid middleware. Worqbench provides high-level services that correspond to the engineering stages in a software development life cycle and they could be adopted as open standards of grid development services.

We have developed a reference implementation of RDT (Section 4.3) that leverages Eclipse as the host IDE. RDT augments Eclipse with functions for developing grid applications. The implementation provides a runtime programmable facility with a set of API and an event mechanism. It allows programmers to automate and scale programming tasks for a large number of resources in an effective manner. The current implementation includes an embedded interpreter for the Groovy programming language. The programmers can utilize the Eclipse and RDT framework APIs by writing scripts and event callback methods in Groovy. We have also implemented the RDT multi-view code editor that simplifies the development of multi-platform applications.

The next chapter presents the demonstrations of GridDebug, Worqbench, RDT, and the overall ISENGARD software infrastructure.

Chapter 5

System Demonstration

This chapter presents a series of four case studies to demonstrate the applicability of the ISENGARD software infrastructure. In particular, we demonstrate the use and power of GridDebug, Worqbench, and RDT in case studies that reflect common situations faced by grid software developers and users. These situations correspond to the stages in the implementation and testing phase of a software development life cycle (Section 2.1).

The case studies cover a succession of activities such as managing grid resources; writing source code files; and executing and debugging grid applications. Importantly, the chapter doesn't set out to measure the quality of our solution using any formal usability metrics. This is because there is such a lack of software in this domain that it would be difficult to perform any real comparison. Further, it is not even clear what usability metrics would be appropriate, and it is difficult to conceive of a fair and effective testing strategy. Finally, the improvement offered by integrated development environments is very subjective, and relies on a user's predisposition to these types of tools. As a result, the goal of the chapter is to demonstrate that our overall architecture works and that the proof-of-concept implementations provide workable solutions to the problems identified in Chapter 1.

The first three case studies are concerned with a small application (approximately 1,400 lines of C code) that implements the Jacobi method [Bronshtein et al., 2007]. The code size and simplicity allow us to present detailed

demonstrations of ISENGARD components. The three case studies are in successive order: preparing grid resources and resource sets (case study 1); developing the software with the multi-view code editor and automation scripts (case study 2); and executing and debugging the software on multiple resources (case study 3). Although it is possible to write grid software without being concerned about the target platforms in advance, we decided to present the first case study before the second one. It gives us the opportunity to describe various methods of accessing ISENGARD and its management functions.

The fourth case study shows the applicability of ISENGARD to larger scientific projects. It concerns a substantial software package for molecular dynamics simulation, called GROMACS [GROMACS, 2009], consisting of approximately 300,000 lines of C code across 700 source files. The case study briefly demonstrates the editing of GROMACS source files with the multi-view code editor; compiling GROMACS on a remote grid platform; and launching the test application that is provided in the GROMACS distribution. The case study does not repeat the details that are presented in the first three studies, but highlights that our techniques appear to scale to non-mockup applications.

This chapter is organized as follows: Section 5.1 provides the details of the grid testbed and software that are used to conduct the case studies. Section 5.2 demonstrates alternative ways of accessing ISENGARD and managing grid resources, sets, projects, and sessions. Section 5.3 presents a case study on the multi-view code editor and examples of using RDT to automate a variety of programming tasks. Section 5.4 demonstrates the software execution and debugging functions offered by the ISENGARD infrastructure. Section 5.5 presents a case study on using the ISENGARD infrastructure to edit, build, and run GROMACS. Finally, Section 5.6 gives a summary of the chapter.

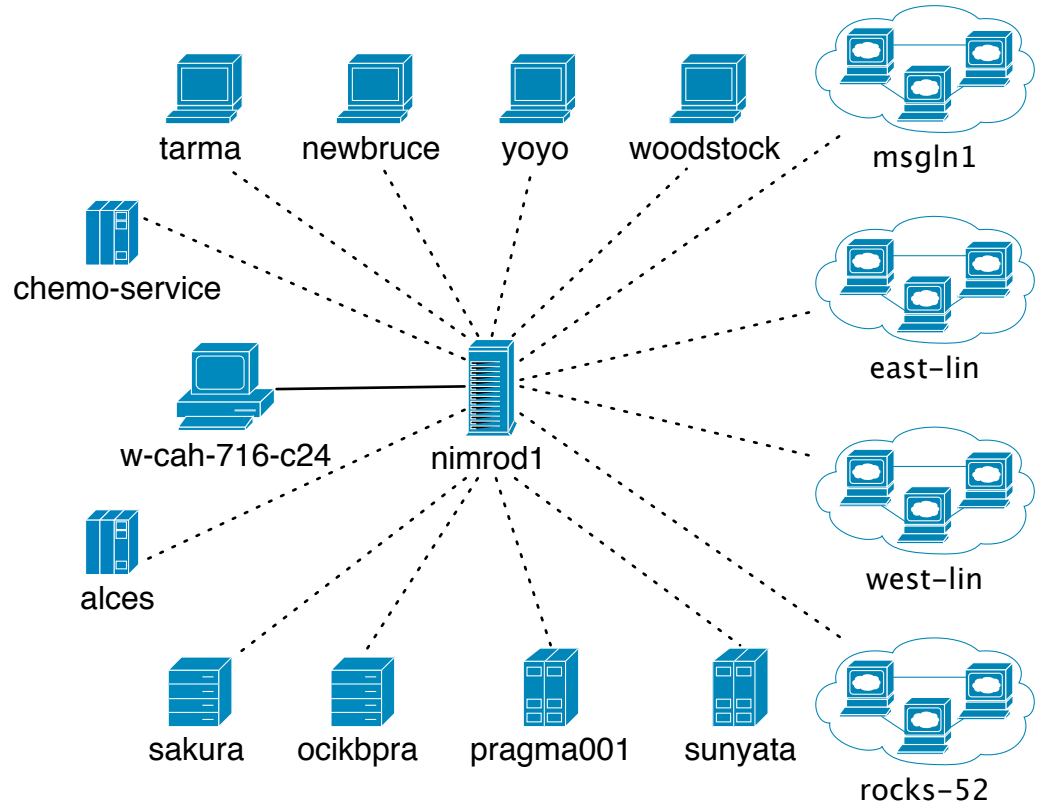


Figure 5.1: Network layout of the testbed

5.1 Grid Testbed and Software

To perform the case studies, we have developed a grid testbed that consists of various resources within several administrative domains. Figure 5.1 shows the network layout of the grid testbed and the resource details are given in Table 5.1. The testbed consists of eight resources at Monash University and eight resources in different organizations and countries. We deliberately chose the resources to represent a range of operating systems, hardware architectures, configurations, and resource types.

We used the machine `w-cah-716-c24` as "the programmer's desktop" to run the IDE (Eclipse RDT) and develop grid software. Worqbench was installed and run on `nimrod1` whilst GridDebug was deployed on the Globus gatekeepers of two clusters, `east-lin` and `west-lin`. The testbed comprises four compute clusters at Monash University (Australia), Deakin University

| Country | Organization | Hostname | OS | CPU | Memory | Type |
|-------------|-------------------------|--|-------------------------|---------------------------------------|--------|-----------------------|
| Australia | Monash Uni. | w-cah-716-c24.infotech. monash.edu.au | Ubuntu Linux 8.04 | Dual-Core Intel Pentium 4 (2.8GHz) | 1GB | Eclipse RDT Client |
| Australia | Monash Uni. | nimrod1.infotech. monash.edu.au | Fedora Core Linux 4 | Dual-Core Intel Pentium 4 (3.0GHz) | 1GB | Workbench Server |
| Australia | Monash Uni. | tarma.csse.monash.edu.au | Solaris 9 | UltraSparc-III (440MHz) | 0.5GB | SSH |
| Australia | Monash Uni. | newbruce.csse.monash.edu.au | Solaris 10 | UltraSparc T1 (1GHz) | 2GB | SSH |
| Australia | Monash Uni. | yoyo.its.monash.edu.au | FreeBSD 5.4 | Quad-Core Intel Xeon (2.0GHz) | 2GB | SSH |
| Australia | Monash Uni. | woodstock.local | Mac OS X 10.5 | PowerPC G5 (1.6GHz) | 2GB | SSH |
| Australia | Monash Uni. | msgln1.its.monash.edu.au | Scientific Linux 5.1 | 2 x Dual-Core AMD Opteron (2.6GHz) | 64GB | Cluster (GT4+SGE) |
| Australia | Monash Uni. | east-lin.enterprisegrid.edu.au | CentOS Linux 5.2 | Quad-Core Intel Xeon (1.6GHz) | 8GB | Cluster (GT4+SGE) |
| Australia | Deakin Uni. | west-lin.enterprisegrid. edu.au | CentOS Linux 5.2 | Quad-Core Intel Xeon (1.6GHz) | 8GB | Cluster (GT4+SGE) |
| USA | SDSC | rocks-52.sdsc.edu | CentOS Linux 4.2 | Dual-Core Intel Xeon (2.4GHz) | 4GB | Cluster (GT2+SGE) |
| Japan | AIST | sakura.hpec.jp | CentOS Linux 5.0 | 2 x AMD Opteron (2.0GHz) | 2GB | GT4 |
| Switzerland | University of Zurich | ocikbpra.unizh.ch | CentOS Linux 4.6 | Dual-Core Intel Xeon (2.8GHz) | 2GB | GT4 |
| Taiwan | ASGC | pragma001.grid.sinica.edu.tw | Scientific Linux 3.0 | Dual-Core Intel Pentium D (3.0GHz) | 2GB | GT2 |
| Thailand | ThaiGrid | sunyata.thaigrid.or.th | CentOS Linux 4.4 | Quad-Core Intel Xeon (2.8GHz) | 4GB | GT2 |
| Germany | TU Dresden | chemo-service.zih.tu- dresden.de | SUSE Linux SLES 10 | Dual-Core AMD Opteron (2.5GHz) | 2GB | UNICORE |
| Poland | ICM | alces.icm.edu.pl | Fedora Core Linux 6 | 4 x Dual-Core AMD Opteron (2.2GHz) | 16GB | UNICORE |

Table 5.1: Testbed resources

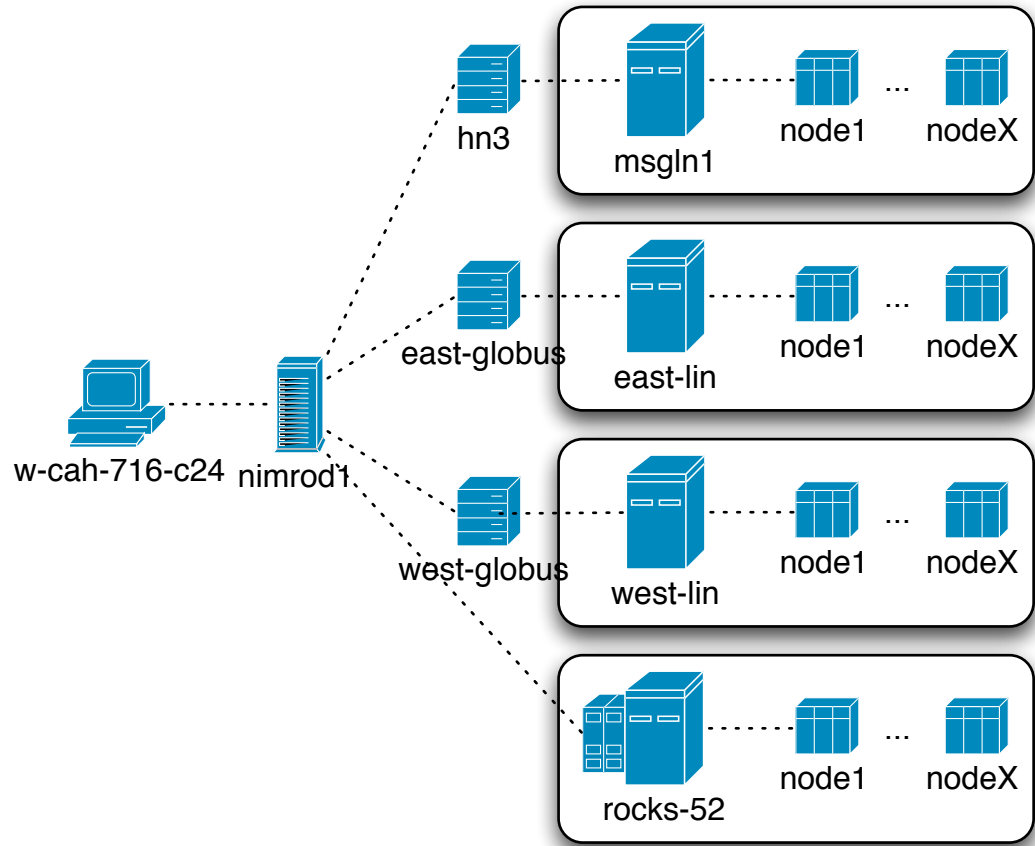


Figure 5.2: Topology of the testbed clusters

(Australia), and San Diego Supercomputer Center (USA). Figure 5.2 shows the topology of the testbed clusters.

The Monash Sun Grid is a high-performance compute cluster at Monash University designed to handle data-intensive applications [MSG, 2008]. The cluster head node is `msgln1` and it utilizes a virtual machine (`hn3`) to handle Globus job submissions. The East and West clusters are part of the Enterprise Grid, a research facility by a consortium of Australian universities, designed to facilitate computer science research in distributed computing [Enterprise Grid, 2008]. The cluster head nodes are `east-lin` and `west-lin` respectively; and the GT4 jobs are submitted through virtual hosts (`east-globus` and `west-globus`). Rocks-52 [Rocks-52, 2008] is a cluster of

15 Linux nodes at San Diego Supercomputer Center (SDSC) for development and testing of grid software in PRAGMA (Pacific Rim Application and Grid Middleware Assembly) [PRAGMA, 2008]. The head node of the cluster, **rocks-52**, also acts as the Globus gatekeeper. All of the four clusters in our testbed utilize the Sun Grid Engine [SGE, 2008] as the job scheduler software.

To demonstrate the software development functions in the first three case studies, we used a small C application provided by Chan and Abramson [2005]. The program is an iterative linear algebra solver known as the Jacobi method [Bronshtein et al., 2007]. The fourth case study concerns GROMACS [GROMACS, 2009], which is a versatile software package that performs molecular dynamics simulation for systems of hundreds to millions of particles. It is primarily designed for biochemical molecules like proteins and lipids; and it has been used extensively in the distributed computing project Folding@Home [Folding@Home, 2009]. GROMACS is written in C and it is multi-platform with specialized low-level instruction sets for different processors such as Opteron and Xeon.

5.2 Case Study 1: Client Access and Management

The first case study concerns client access and management functions. Specifically, the aim of the case study is to demonstrate various methods to access Worqbench and RDT functions for grid software development. The access methods range from a simple and ubiquitous web interface to a set of API functions. We also describe and demonstrate the process of initializing access to grid resources; setting up appropriate resource sets; and managing application projects and sessions. These activities need to be performed before the user can take advantage of various ISENGARD services.

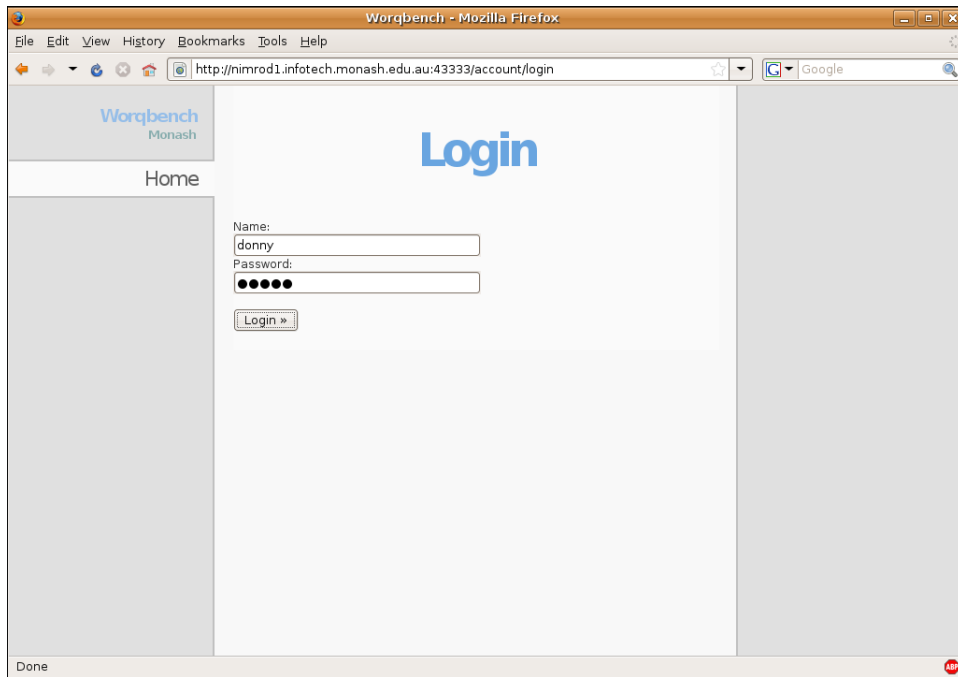


Figure 5.3: Worqbench login page

5.2.1 Client Access

Worqbench can be accessed in several different ways, one of which is through a standard web browser. The Worqbench web portal, as shown in Figures 5.3, 5.4, and 5.5, provides a simple and user-friendly web interface for users to manage resources, sets, projects, sessions, and tasks. Figure 5.3 displays the Worqbench login page whilst Figure 5.4 shows the page for an administrator to manage system resources. Figure 5.5 displays the management page for user resources for the user "donny". The user information is shown in the left sidebar which also contains links to web management pages for user-specific Worqbench entities.

As described in Section 2.9.1, the Worqbench portal provides uniform and simple interfaces for accessing grid resources and jobs. It is accessible via web browsers without the need to install specialized software.

However, the simplicity of a web portal and HTML forms could be a hindrance compared to a proper development environment. Consequently, in addition to a web browser, Worqbench can be accessed through an IDE with

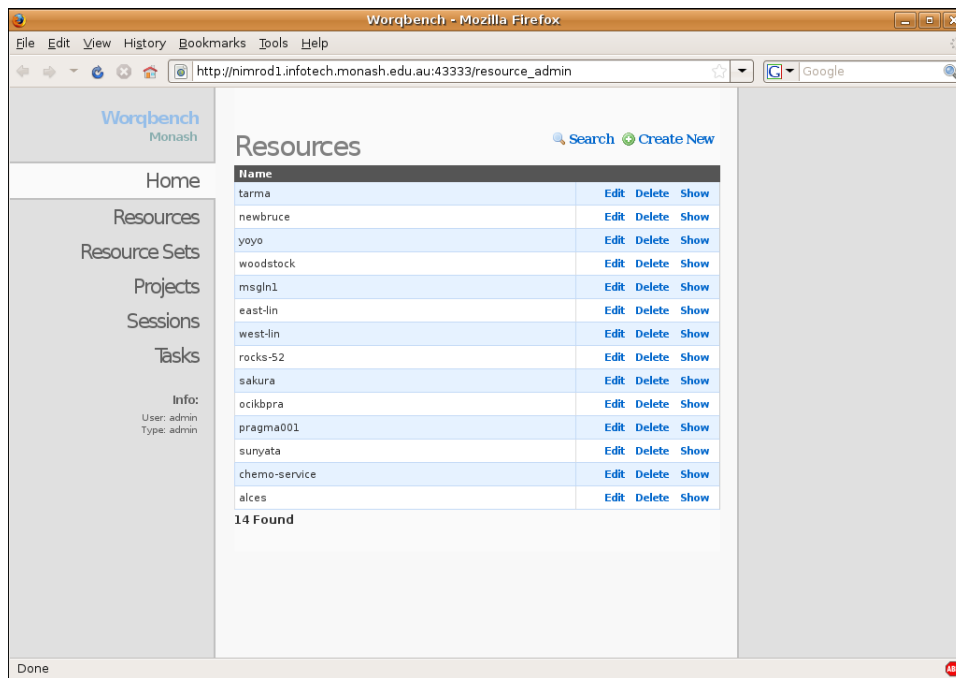


Figure 5.4: Management page for system resources

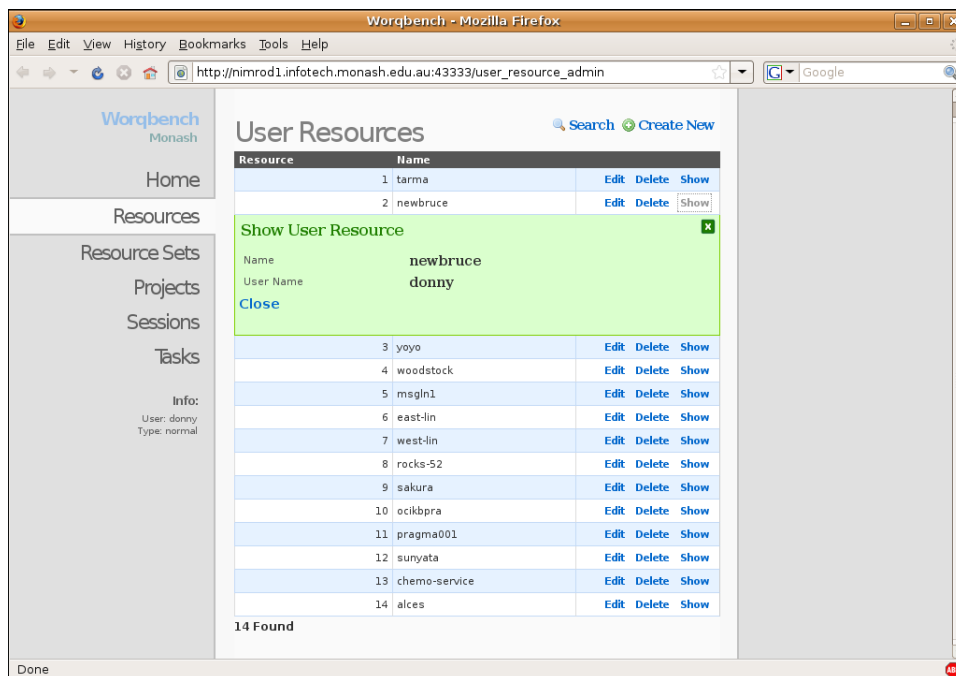


Figure 5.5: Management page for user resources

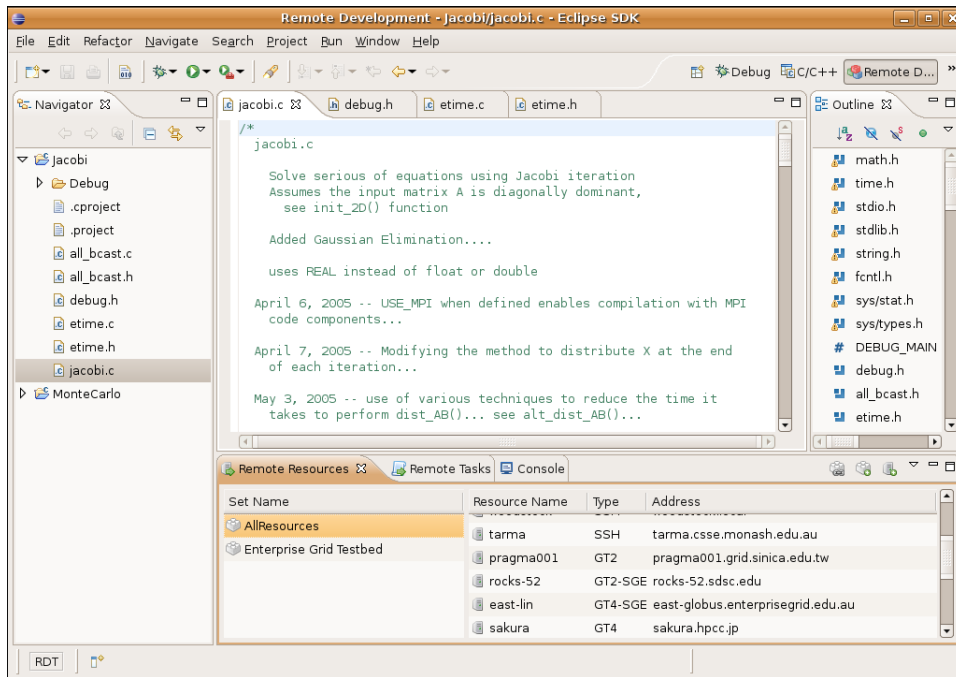


Figure 5.6: Eclipse RDT and its Remote Development perspective

proper plug-ins as exemplified by Eclipse RDT. This IDE provides a versatile and effective environment to utilize Worqbench services. Figure 5.6 shows the Eclipse RDT and its Remote Development perspective that organizes windows and editors related to grid application development. In the figure, the left, centre, and right panes display the Eclipse project navigator, editor, and code outline respectively. The bottom pane shows the Remote Resources view that allows programmers to create, edit, destroy, and manage remote resources and sets. The Worqbench connection details are specified through the Eclipse preferences dialog box as shown in Figure 5.7.

The third access mechanism is through API methods that allow users to leverage the services directly without being limited by the implementation of IDE plug-ins. Worqbench offers web service interfaces that are independent of the programming language and platform. The API provides a flexible and powerful way to utilize Worqbench services, for example, to perform "offline" or *unattended* operations through scripts or `cron` jobs. Figure 5.8 displays an example of accessing Worqbench from an Interactive Ruby Shell

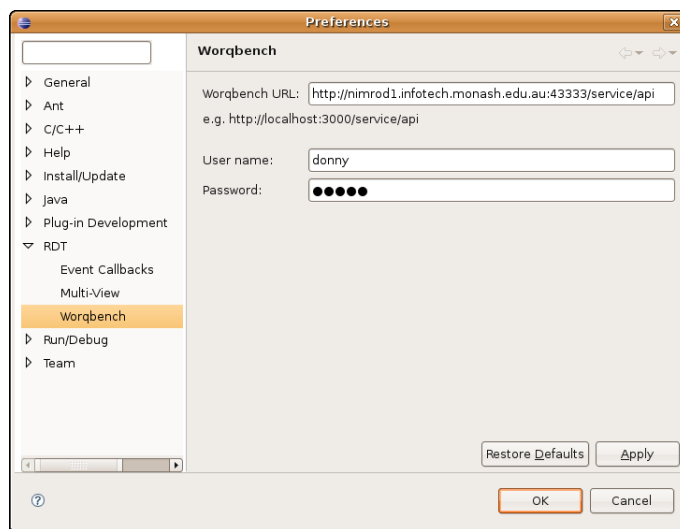


Figure 5.7: Eclipse preferences window for Worqbench

(IRB) session. In the figure, Ruby statements are preceded by `>>` whilst the evaluation results are preceded by `=>`. The code in Figure 5.8 can be easily expanded into complete administration scripts. For example, scripts to check for unavailable resources or build dynamic resource sets based on predefined conditions.

Unlike other grid tools, Worqbench exposes the grid development services and management functions (discussed in Section 5.2.2) using a variety of access methods. Each method has advantages to fit different usage requirements and scenarios. The web portal provides a user-friendly and simple interface from a web browser; and the IDE interface provides an effective development environment specifically suited to the needs of grid programmers. The API library allows the services to be accessed programmatically and they can be extended to operate with other grid software or customized as high-level development functions.

The next section (Section 5.2.2) discusses management functions for various Worqbench entities.

```

donny@w-cah-716-c24: ~
File Edit View Terminal Tabs Help
donny@w-cah-716-c24:~$ irb --simple-prompt
>> require 'xmlrpc/client'
=> true
>> server = XMLRPC::Client.new2("http://nimrod1.infotech.monash.edu.au:43333/service/api")
=> #<XMLRPC::Client:0xb7d416f8 @auth=nil, @cookie=nil, @create=nil, @port=43333,
  @http=#<Net::HTTP nimrod1.infotech.monash.edu.au:43333 open=false>, @proxy_host=
nil, @http_last_response=nil, @parser=nil, @timeout=30, @path="/service/api", @
password=nil, @http_header_extra=nil, @use_ssl=false, @host="nimrod1.infotech.mo
nash.edu.au", @user=nil, @proxy_port=nil>
>> key = server.call("admin.GetKey", "donny", "donny")
=> "ce34ca2b9c4ef4cf336159966c534cefe84596b4-2"
>> number_of_my_resources = server.call("resource.List", key).length
=> 14
>> name_of_my_first_resource = server.call("resource.List", key)[0]["name"]
=> "tarma"
>> list_of_worqbench_users = server.call("admin.GetUsers", key).collect {|u| u["
name"]}
=> ["admin", "donny"]
>> list_of_system_resources = server.call("admin.GetResources", key).collect {|r
| r["name"] + "/" + r["kind"]}
=> ["tarma/SSH", "newbruce/SSH", "yoyo/SSH", "woodstock/SSH", "msgln1/GT4-SGE",
"east-lin/GT4-SGE", "west-lin/GT4-SGE", "rocks-52/GT2-SGE", "sakura/GT4", "ocikb
pra/GT4", "pragma001/GT2", "sunyata/GT2", "chemo-service/UNICORE", "alces/UNICOR
E"]
>> exit
donny@w-cah-716-c24:~$

```

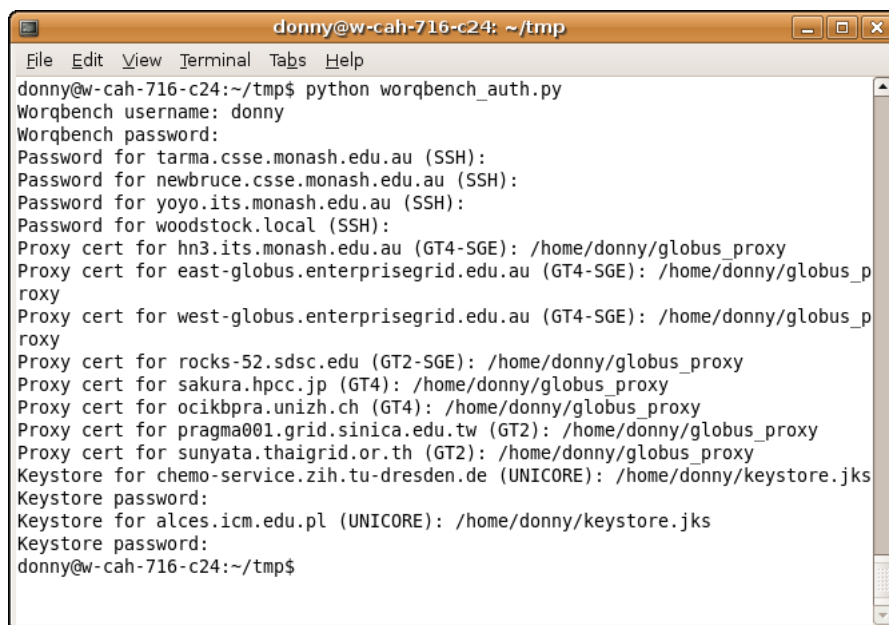
Figure 5.8: Accessing Worqbench from an IRB session

5.2.2 Entity Management

As described in Sections 3.4.2 and 3.5.1.1, there are a number of participating "objects" or entities involved in grid application development. The ISENGARD services work with these entities and they need to be predefined before we can take advantage of the ISENGARD infrastructure.

Worqbench provides various management functions for resources, sets, projects, and sessions. They are accessible through the same access methods as described in the previous section. They can be employed from the web portal or the IDE (Eclipse RDT); or utilized from scripts. This versatility is advantageous since it can cater for a range of different user scenarios. For example:

- A user wants to initialize and access several resources that have different authentication mechanisms. This process may need to be carried out for several times as new resources are added to the system. By using the API library and a script, the task can be repeated and accomplished



```

donny@w-cah-716-c24: ~/tmp
File Edit View Terminal Tabs Help
donny@w-cah-716-c24:~/tmp$ python worqbench_auth.py
Worqbench username: donny
Worqbench password:
Password for tarma.csse.monash.edu.au (SSH):
Password for newbruce.csse.monash.edu.au (SSH):
Password for yoyo.its.monash.edu.au (SSH):
Password for woodstock.local (SSH):
Proxy cert for hn3.its.monash.edu.au (GT4-SGE): /home/donny/globus_proxy
Proxy cert for east-globus.enterprisegrid.edu.au (GT4-SGE): /home/donny/globus_p
roxy
Proxy cert for west-globus.enterprisegrid.edu.au (GT4-SGE): /home/donny/globus_p
roxy
Proxy cert for rocks-52.sdsc.edu (GT2-SGE): /home/donny/globus_proxy
Proxy cert for sakura.hpcc.jp (GT4): /home/donny/globus_proxy
Proxy cert for ocikbp.ra.unizh.ch (GT4): /home/donny/globus_proxy
Proxy cert for pragma001.grid.sinica.edu.tw (GT2): /home/donny/globus_proxy
Proxy cert for sunyata.thaigrid.or.th (GT2): /home/donny/globus_proxy
Keystore for chemo-service.zih.tu-dresden.de (UNICORE): /home/donny/keystore.jks
Keystore password:
Keystore for alces.icm.edu.pl (UNICORE): /home/donny/keystore.jks
Keystore password:
donny@w-cah-716-c24:~/tmp$

```

Figure 5.9: Execution of a script to initialize access to user resources

in an efficient manner

To illustrate this, Listing 5.1 shows a Python script that initializes authentications to user resources. The script sets up the connection to Worqbench (lines 5-9) and retrieves a list of system resources (line 12). It then builds a hash table of system resources (line 13) from the array list. The script then retrieves a list of user resources (line 15) and iterates over it (lines 17-41). If the user resource is an SSH resource, the script asks for a password (line 24) and edits the `user_password` attribute of the resource (line 25). Similarly, the script asks for a proxy certificate for a Globus resource (lines 27-31); and a keystore file and its password for a UNICORE resource (lines 33-39). Finally, the script prints a message if there are any errors (line 41). The execution output of the script is displayed in Figure 5.9. A Worqbench user can perform the same activity through the web portal although the script can accomplish it in a quick and efficient manner.

- An e-Scientist whose laptop is not readily available, needs to immedi-

```

1 import xmlrpclib
2 import getpass
3 import sys
4
5 server = xmlrpclib.Server('http://nimrod1.infotech.monash.↵
    edu.au:43333/service/api')
6 sys.stdout.write("Worqbench username: ")
7 worq_username = sys.stdin.readline().rstrip()
8 worq_password = getpass.getpass("Worqbench password: ")
9 key = server.admin.GetKey(worq_username, worq_password)
10
11 system_resources = {}
12 for sys_res in server.admin.GetResources(key):
13     system_resources[sys_res['id']] = sys_res
14
15 my_resources = server.resource.List(key)
16
17 for my_res in my_resources:
18     sys_res = system_resources[my_res['resource_id']]
19     kind = sys_res['kind']
20     address = sys_res['address']
21     isSuccess = False
22
23     if kind.startswith('SSH'):
24         password = getpass.getpass("Password for %s (%s): " % (↵
            address, kind))
25         isSuccess = server.resource.Edit(key, my_res['name'], '↵
            user_password', password)
26
27     elif kind.startswith('GT'):
28         sys.stdout.write("Proxy cert for %s (%s): " % (address, ↵
            kind))
29         cert_path = sys.stdin.readline().rstrip()
30         cert_data = xmlrpclib.Binary(open(cert_path, 'r').read()↵
            )
31         isSuccess = server.resource.Edit(key, my_res['name'], '↵
            user_certificate', cert_data)
32
33     elif kind.startswith('UNICORE'):
34         sys.stdout.write("Keystore for %s (%s): " % (address, ↵
            kind))
35         key_path = sys.stdin.readline().rstrip()
36         key_password = getpass.getpass("Keystore password: ")
37         key_data = xmlrpclib.Binary(open(key_path, 'r').read())
38         isSuccess = server.resource.Edit(key, my_res['name'], '↵
            user_certificate', key_data)
39         isSuccess &= server.resource.Edit(key, my_res['name'], '↵
            user_password', key_password)
40
41 if not isSuccess: print "ERROR: %s (%s)" % (address, kind)

```

Listing 5.1: A Python script to initialize access to user resources

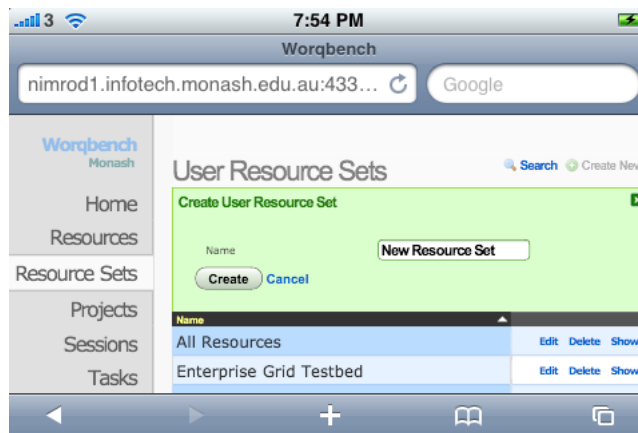


Figure 5.10: Management page for user resource sets

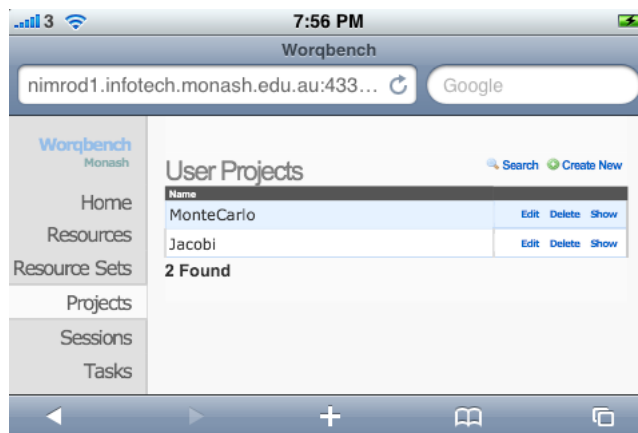


Figure 5.11: Management page for user projects

ately create a new resource set and re-execute a predefined project. The e-Scientist, who has access to an iPhone, can login to the Worqbench web portal and manage the relevant entities (Figures 5.10 and 5.11) to achieve the objective. Figure 5.10 shows the web portal page for managing user resource sets whilst Figure 5.11 shows the corresponding page for user projects. Clearly, this ubiquitous access to grid resources and projects is beneficial for users who need a simple web interface or who may not have IDEs installed on their workstations.

- A programmer, who is writing a grid application, wants to test the software on several resources quickly. The programmer can use Eclipse

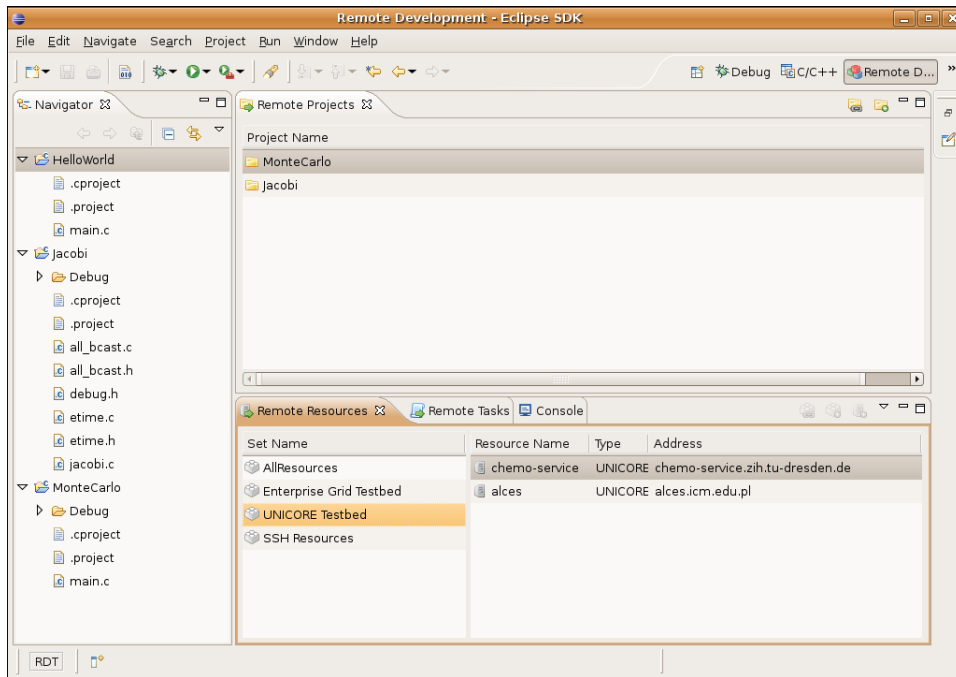


Figure 5.12: Remote Projects and Remote Resources views

RDT and its Remote Resources view (the bottom pane in Figure 5.12) to define new resources and resource sets; and deploy the software project. It allows a tight integration between the software development process and the execution platforms. The top pane in Figure 5.12 shows the Remote Projects view that allows users to create, edit, destroy, and manage remote projects. In Figure 5.12, the project navigator (the left pane) lists local Eclipse projects where two of them ("MonteCarlo" and "Jacobi") are synced with the corresponding Worqbench remote projects.

Additionally, Worqbench provides management functions for user sessions. As discussed in Sections 3.4.2 and 3.5.1.1, a user session represents a clear and definite linkage between the application project and the target platform. It allows Worqbench and Eclipse RDT to perform operations, such as software compilation and execution, on users' behalf unambiguously. The sessions can be managed through the Worqbench web portal as displayed in Figure 5.13. In the figure, a session called

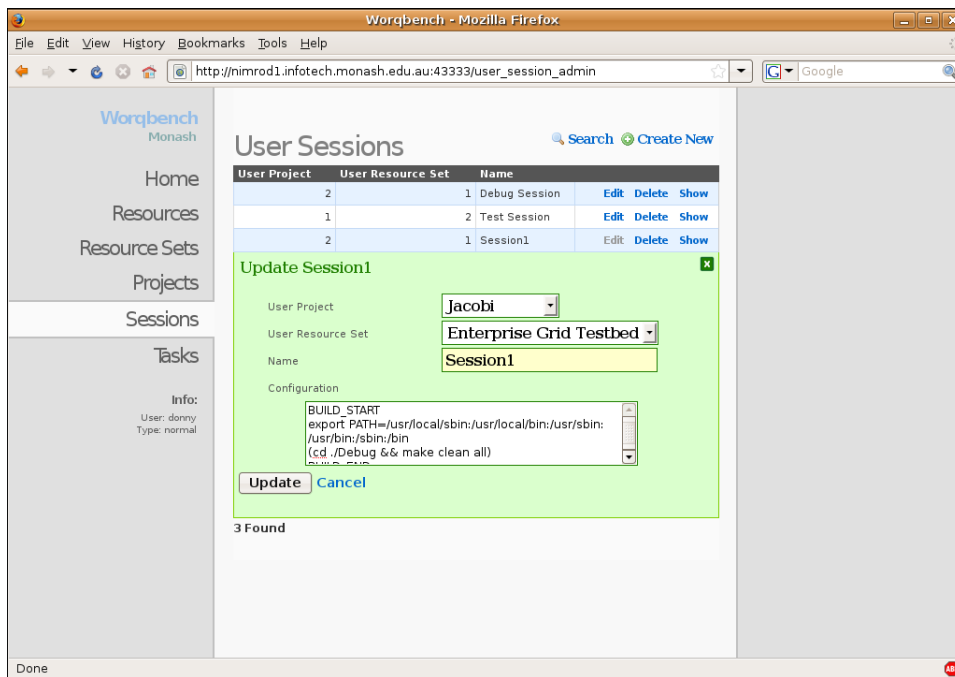


Figure 5.13: Management page for user sessions

”Session1” is associated with the ”Jacobi” project and the ”Enterprise Grid Testbed” set. The figure also shows the session configuration which stores information such as compilation and execution commands.

The aforementioned user scenarios reflect plausible situations faced by grid users and developers. Worqbench provides management functions through various access methods that can cater for a range of different scenarios. More importantly, users are not limited to only one access method. Thus, they can employ one mechanism or more, as desired, according to their current situations and requirements.

5.3 Case Study 2: IDE Scripting and Code Editing

After the target platforms and entities have been defined, developers can employ the Eclipse RDT to write grid applications. Section 2.3 has described

the advantages of using an IDE to develop applications and support the full software development life cycle. The IDE functions are further improved by Eclipse RDT, as discussed in Sections 3.5 and 4.3. Eclipse RDT offers new features that simplify and automate a variety of grid programming activities.

This case study concerns IDE scripting and source code editing. In particular, there are two objectives. First, we want to show and discuss the use of scripting to automate various programming tasks and the IDE. Second, we want to demonstrate the RDT multi-view code editor for writing multi-platform software.

5.3.1 Automation Scripts

In addition to the point-and-click interaction that is common among IDEs, Eclipse RDT allows developers to write scripts to interact and drive the IDE. The use of a high-level scripting language to drive tools is advantageous since the developers can customize the tool functions to cater for variations in their particular grid infrastructure. Moreover, it allows the developers to automate many tedious grid programming tasks for a large number of resources. Certainly, this mode of interaction with an IDE is more efficient and simpler for repetitive tasks compared to the point-and-click interaction.

The primary way to "script" the IDE is through the Groovy Shell view that is shown in Figure 5.14. The view has two text boxes: an input box to write a Groovy script and an output box to display the script's execution output. The Groovy Shell view allows the user to save the script to a text file or load a pre-written Groovy script. Figure 5.14 shows a script that calls Eclipse API to display a message box. In this section, we list two other examples of Eclipse RDT automation scripts.

The first example demonstrates that the Eclipse RDT runtime programmable facility works with the built-in Eclipse objects, specifically its project data structures. As described in Section 4.3.1, Eclipse RDT supports two project abstractions: the standard Eclipse IProject for local projects and the RDT IProject for grid projects. The RDT IProject model acts as a thin wrapper around the Eclipse IProject and implements the necessary data

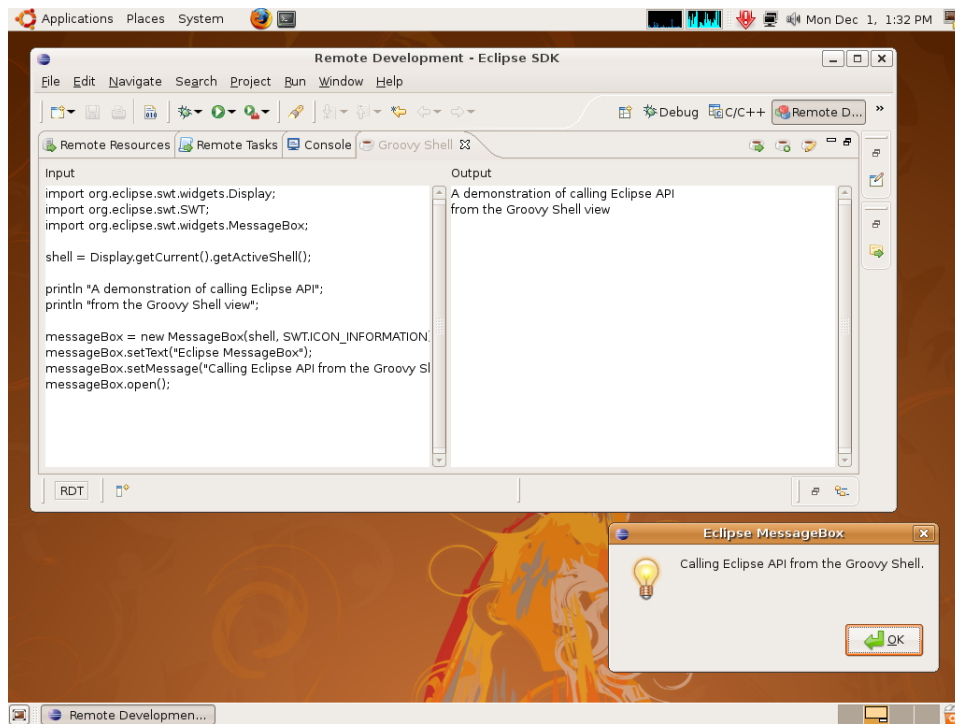


Figure 5.14: Groovy Shell view

```

1 import org.eclipse.core.resources.IProject;
2 import org.eclipse.core.resources.ResourcesPlugin;
3
4 workspaceRoot = ResourcesPlugin.getWorkspace().getRoot();
5 localProjects = workspaceRoot.getProjects();
6 remoteProjects = ProjectManager.getProjects();
7
8 println "List of local Eclipse projects:\n";
9 for (project in localProjects) {
10     name = project.getName();
11     println name;
12     cmdChangeProjectImage(name, "BLUE");
13     cmdChangeProjectText(name, "LOCAL"); }
14
15 println "\n\nList of Worqbench projects:\n"
16 for (project in remoteProjects) {
17     name = project.getProjectName();
18     println name;
19     cmdChangeProjectImage(name, "RED");
20     cmdChangeProjectText(name, "GRID"); }

```

Listing 5.2: A Groovy script to list software projects

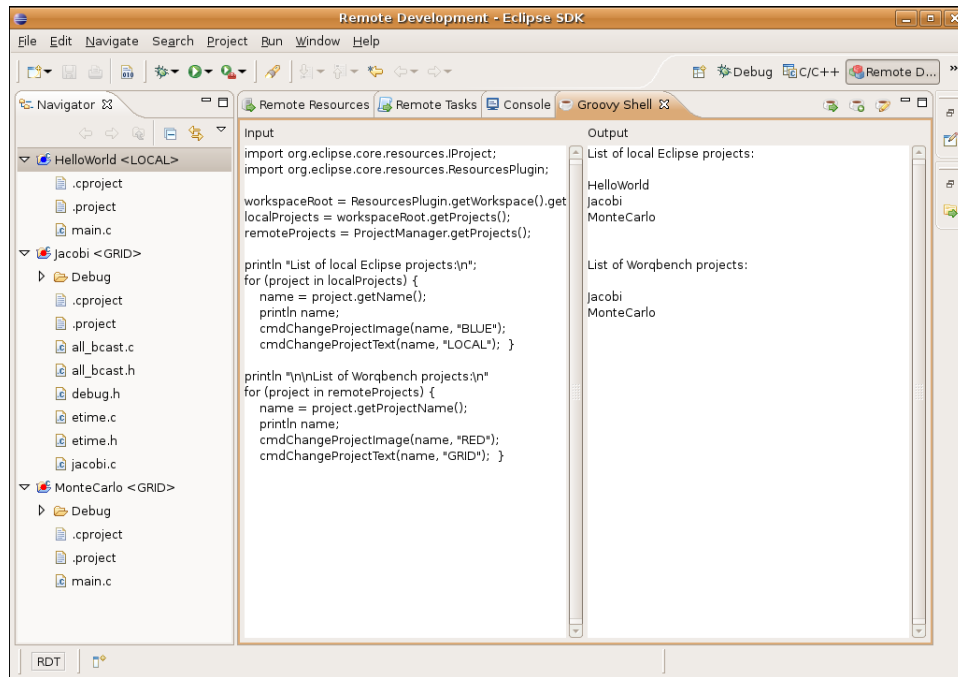


Figure 5.15: Navigator and Groovy Shell views

structure as required by RDT and Worqbench. Listing 5.2 shows a Groovy script that lists local Eclipse and RDT/Worqbench projects. It retrieves Eclipse projects in the current workspace by calling the built-in Eclipse API (lines 4-5); and retrieves a list of Worqbench projects by calling the `ProjectManager.getProjects()` method (line 6). The script then marks all Eclipse projects with the text "LOCAL" and the colour blue (lines 9-13); and Worqbench projects with the text "GRID" and the colour red (lines 16-20). Figure 5.15 shows the execution of the script and the project navigator (the left pane) that lists marked projects. The script allows the user to easily distinguish between grid and non-grid projects.

The second automation example concerns grid software projects and resources; and is best illustrated by the following scenario:

Alice has a collection of resources with different operating systems, hardware architectures, and configurations. The resource descriptions are stored in the resource metadata. Alice has been developing several grid applications, each of which requires a cer-

tain minimum amount of RAM (e.g. 512MB).

Alice wants to assemble grid testbeds for the applications where each testbed consists of resources that meet the minimum system requirement for each application.

This hypothetical user scenario is actually realistic and is common among grid developers. It is certainly possible to perform this task outside the IDE by creating shell scripts that check for resource descriptions and group them together. Alternatively, this task can be handled in the IDE to facilitate a quick turnaround between writing and testing phases. However, traditional IDEs that do not have resource and set abstractions may not necessarily accommodate the needs of developers.

Eclipse RDT with its system model and scripting facility assists the user in conducting the aforementioned task effectively. Rather than inspecting every resource and grouping them manually into testbeds, the user can write a Groovy script that does this operation, and is shown in Listing 5.3. The code listing shows a Groovy script that automatically creates resource sets with appropriate resources based on the memory requirement of software projects. It retrieves a list of projects (line 1) and creates a new resource set (line 5) for each project. The script obtains the project's memory requirement based on its tags (lines 7-14). It then checks all resources (line 16) for their memory details based on their resource tags (lines 20-27). If a resource meets the project's memory requirement, it is added to the newly created resource set (lines 32-33).

This kind of automation is powerful and it is further improved by the Eclipse RDT event mechanism that supports script execution based on certain events. The event mechanism and scripts are demonstrated in the next section.

5.3.2 Event Scripts

Eclipse RDT provides an event mechanism based on the framework system model with user-definable callback methods. This mechanism is powerful since it allows users to specify various checks and operations to be performed

```
1 allProjects = ProjectManager.getProjects();
2
3 for (project in allProjects) {
4
5     testbed = SetManager.newSet("Testbed for " + project.↵
6         getProjectName());
7
8     // e.g. "testing requiredMem:512 version:3"
9     projectTags = project.getTags();
10
11    // e.g. "requiredMem:512"
12    projectTagsMatcher = projectTags =~ /requiredMem:(\w+)/;
13
14    // e.g. "512"
15    requiredProjectMem = projectTagsMatcher[0][1];
16
17    allResources = ResourceManager.getResources();
18
19    for (resource in allResources) {
20
21        // e.g. "arch:x86 RAM:1024 os:linux"
22        resourceTags = resource.getTags();
23
24        // e.g. "RAM:1024"
25        resourceTagsMatcher = resourceTags =~ /RAM:(\w+)/;
26
27        // e.g. "1024"
28        resourceRAM = resourceTagsMatcher[0][1];
29
30        memory1 = resourceRAM.toInteger();
31        memory2 = requiredProjectMem.toInteger();
32
33        if (memory1 >= memory2)
34            testbed.addResource(resource.getResourceId());
35    }
```

Listing 5.3: A Groovy script to create resource sets based on projects

when specific events are triggered. To continue with the example given in the previous section, Alice can modify the script in Listing 5.3 into a new `onResourceCreated()` callback method. When she defines a new grid resource, the method will be invoked and it can add the newly created resource to existing grid testbeds depending on its minimum system requirements.

We believe the Eclipse RDT event system increases efficiency and programmers' productivity. By using other IDEs without this automation feature, Alice must check and add the new resource manually to the grid testbeds. Listing 5.4 provides other examples of event callback methods and they are described as follows:

- The method `onFileChanged()` (lines 1-6) is called when a file has been changed by the user. This example reminds them to reload the database when the schema file has been modified. A special variable called `name` that contains the filename is available within the scope of this method. The data type of `name` is Java String. The example prints a warning to the console output when a file with the extension `sql` is changed.
- The method `onResourceAdded()` (lines 8-18) is called when a resource is added to a set. This example warns the user when a Windows resource is added to a Globus Toolkit testbed. Two special variables, `resName` and `setName`, are available within this method. At line 11, we initialize a new variable `vSet` that holds the actual set object and at line 13, we store `vSet`'s tags in the variable `setTags`. We do the corresponding operations with the resource at lines 12 and 14. At line 16, we perform tests on the tags and print a warning if the conditional is true.
- The method `onSetDestroyed()` (lines 20-29) is called when a set is destroyed with a special variable `name` that identifies the set. The example checks all projects for the set's name in the project's tags. If the conditional is true, matched projects are marked with the text "Inspect Me!" which is visible in the Eclipse's project navigator.

The callback methods are defined through the Eclipse preferences window


```
1 def onFileChanged() {
2     println("File: " + name + " has been changed");
3
4     if (name.endsWith(".sql"))
5         println("Reload the database!");
6 }
7
8 def onResourceAdded() {
9     println("Set is: " + setName + " and Resource is: " + ↵
10         resName);
11
12     vSet = SetManager.getSet(setName);
13     vResource = ResourceManager.getResource(resName);
14     setTags = vSet.getTags();
15     resTags = vResource.getTags();
16
17     if (resTags.indexOf("windows") != -1 && setTags.indexOf("↵
18         GT4") != -1)
19         println("Warning: adding a Windows PC to a Globus ↵
20             Toolkit Testbed");
21 }
22
23 def onSetDestroyed() {
24     println("Set: " + name + " has been destroyed");
25
26     allProjects = ProjectManager.getProjects();
27
28     for (project in allProjects) {
29         if (project.getTags().indexOf(name) != -1)
30             cmdChangeProjectText(project.getProjectName(), "↵
31                 Inspect Me!");
32     }
33 }
```

Listing 5.4: Examples of event callback methods

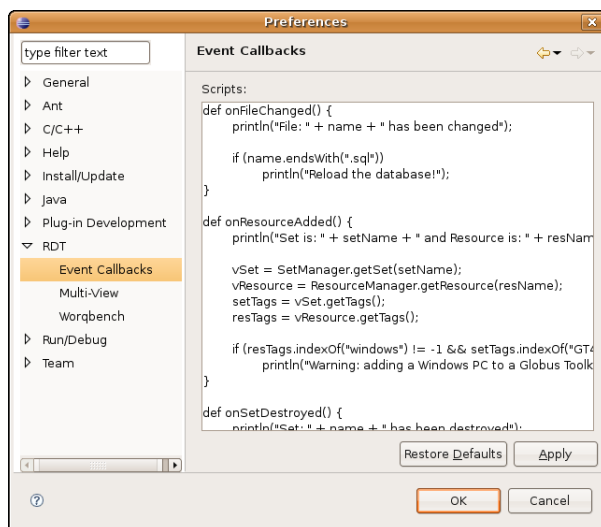


Figure 5.16: Eclipse preferences window for Event Callbacks

as shown in Figure 5.16. When Eclipse RDT is started, the methods will be loaded and registered with the event handlers (Section 4.3.4.1). All built-in Eclipse APIs, data structures, and GUI widgets are callable within the callback methods.

The Eclipse RDT event mechanism presents a significant improvement over traditional IDEs. It allows the development environment to perform automatic check and verification steps on behalf of the programmers. Thus, it allows them to concentrate on the task of writing grid applications. Moreover, by offering a tight integration between the software development process and the execution platforms, RDT enables the programmers to develop grid software in relation to the composition of the grid environment itself.

5.3.3 Code Editing with Multiple Views

In addition to the automation feature, Eclipse RDT also offers an editor that helps developers manage multi-platform software, as discussed in Section 3.5.2.3. It provides a multi-view code editor that simplifies code comprehension by showing only relevant portions of the code and hiding the irrelevant parts. For example, a programmer wants to improve the message passing versions of a scientific application that can be run sequentially or in parallel

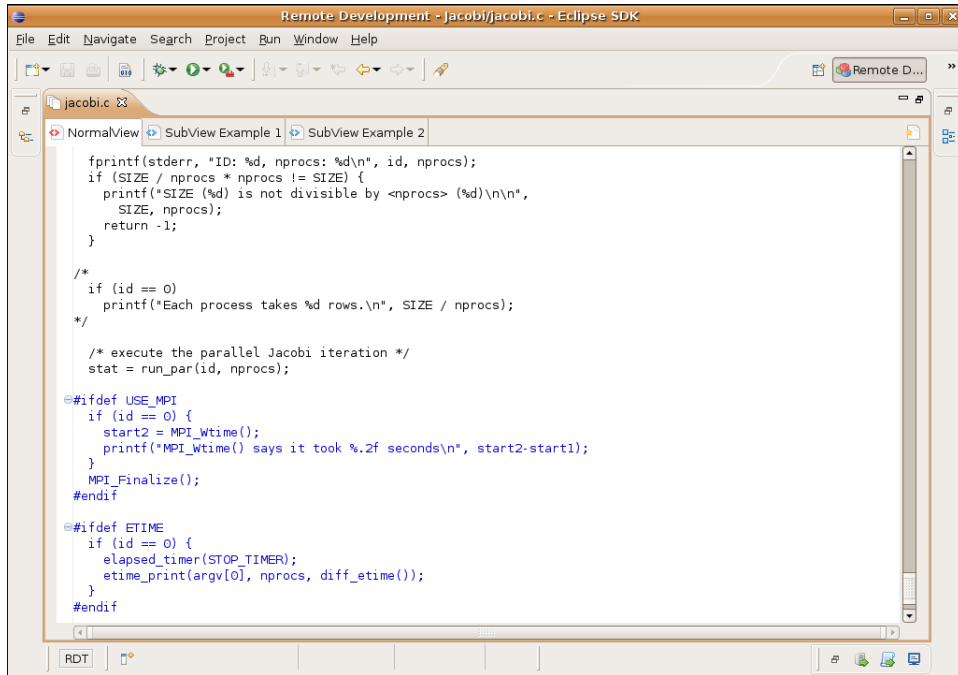


Figure 5.17: Multi-view code editor (normal view)

on a cluster. Using Eclipse RDT, the programmer can open the source files with the multi-view code editor and configure it to show only MPI-specific codes and hide the rest. This allows the programmer to concentrate on the task at hand without being distracted by the serial sections of the code.

The RDT multi-view editor consists of a normal view and multiple sub-views as illustrated in Figure 5.17. The figure shows the multi-view code editor with a normal view (the tab with the red page icon) and two sub-views (the tabs with the blue page icon). The editor partitions a source file into several text sections according to a user-defined regular expression which is specified through the Eclipse preferences window as shown in Figure 5.18.

To illustrate the usage of the editor, consider the Jacobi program introduced in Section 5.1. It was originally written as a sequential program, however, the current version can be compiled with MPI libraries to support parallel execution. It has its own logging and timer functions to record various time related measurements.

Figure 5.17 shows the main project file called `jacobi.c` which has two

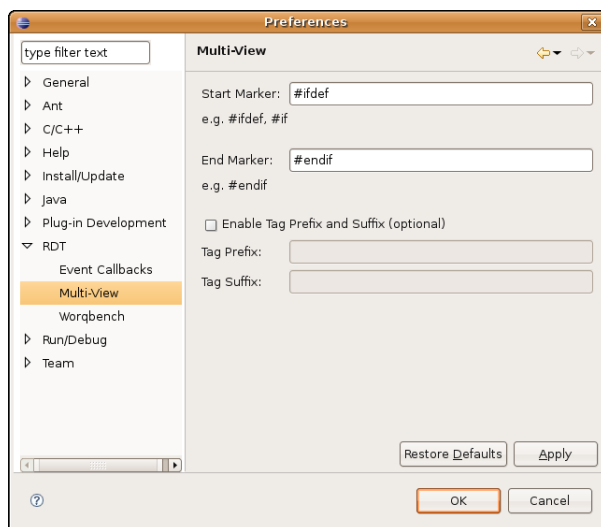


Figure 5.18: Eclipse preferences window for Multi-View



Figure 5.19: New Code View dialog boxes

conditional compilation text blocks (`USE_MPI` and `ETIME`). When a programmer wants to develop the MPI-specific code without being concerned about the time related code, a new sub-view could be created with the `USE_MPI` tag (Figure 5.19 (A)). The new sub-view, as displayed in Figure 5.20, shows only text blocks enclosed by `#ifdef USE_MPI ... #endif` and hides blocks enclosed by other `#ifdef ... #endif`. Similarly, another sub-view could be created with the `USE_MPI` and `ETIME` tags (Figure 5.19 (B)); and it shows only MPI-specific and time related text blocks (Figure 5.21). Notice that the text blocks that are not enclosed by `#ifdef ... #endif` are displayed as they are in the sub-views. In addition, the `#ifdef ... #endif` markers are not shown in the sub-views.

The editor works with a single instance of the file and multiple "in memory" data structures for the sub-views. The programmer can edit the source

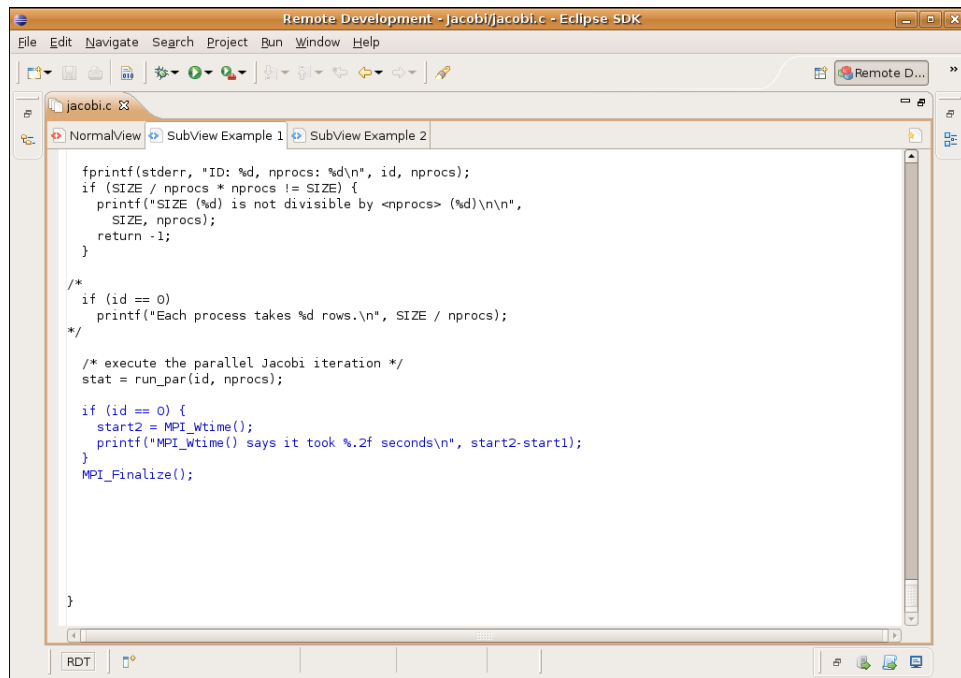


Figure 5.20: Multi-view code editor (sub-view example 1)

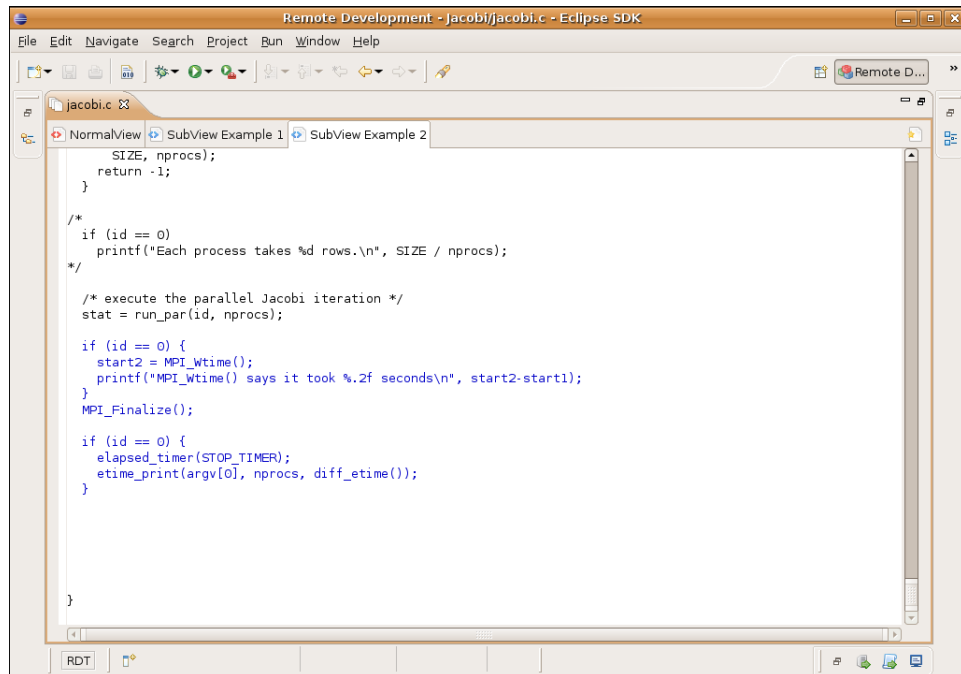


Figure 5.21: Multi-view code editor (sub-view example 2)

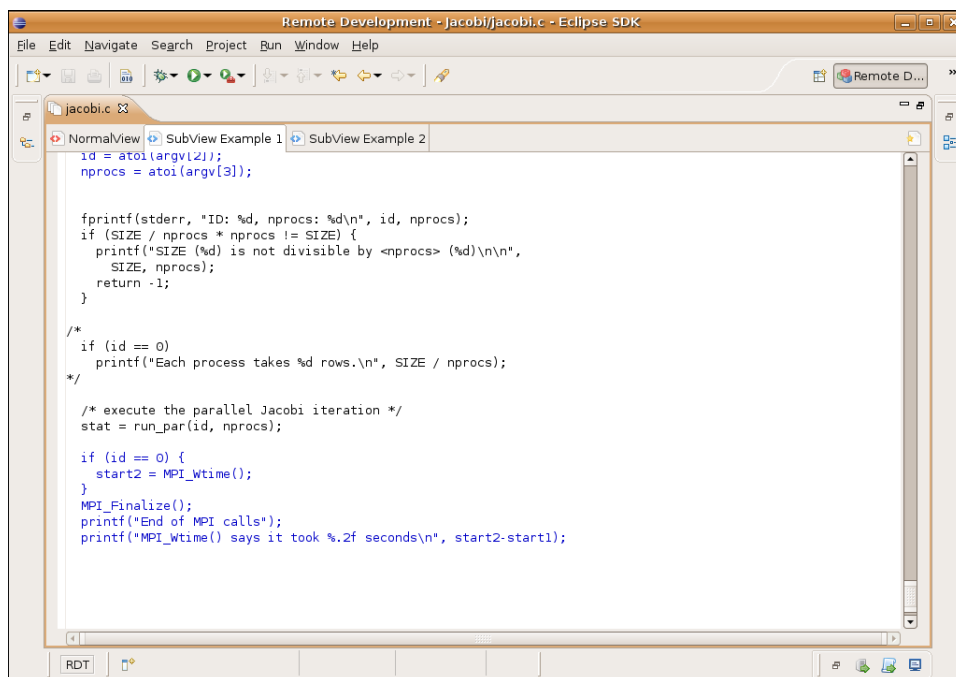


Figure 5.22: Code editing (sub-view example 1)

text in a view and the changes will be saved in the file. It avoids the problem of concurrency by allowing only one view to be active and editable at one time.

The next two figures (Figures 5.22 and 5.23) show an example of using the multi-view editor to edit source code. The programmer edits the source file in "SubView Example 1" (Figure 5.20). When text lines are modified or added to the blue coloured region (Figure 5.22) and the view is switched to "NormalView", the lines will be shown to be enclosed by `#ifdef USE_MPI ... #endif` (Figure 5.23). This example shows that code editing in a view is also reflected in other views.

Writing grid applications for a highly heterogeneous and distributed infrastructure is often complex and error prone since the infrastructure may introduce variations in source code due to resource heterogeneity. To alleviate this, Eclipse RDT provides the multi-view code editor that simplifies code comprehension of multi-platform software. It offers a significant advantage compared to a traditional code editor with a single code view. More

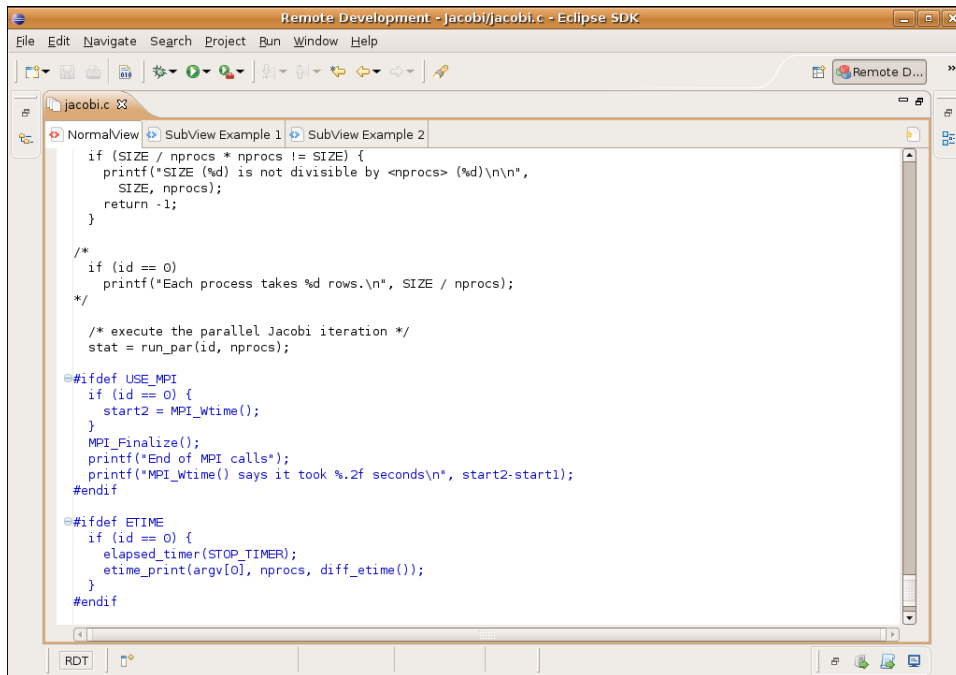


Figure 5.23: Code editing (normal view)

importantly, the multi-view code editor does not add extra information to the files nor modify them with the exception of the editing performed by the programmer. As a result, the programmer is not tied to the multi-view editor and can still use other tools to edit the files.

5.4 Case Study 3: Software Execution and Debugging

After the grid software has been written, it is executed on the target platforms. In order to locate errors, the software may need to be launched under the control of a debugger. In the grid environment, software execution and debugging is further complicated by issues such as job scheduling and access restrictions across a hierarchy of resources. As described in Chapter 3, ISENGARD has been designed to accommodate these non-trivial issues.

This case study discusses the launching stage of grid applications. Specif-

ically, the goal of the study is to demonstrate and analyze grid software execution and debugging performed by using GridDebug, Worqbench, and RDT. It also discusses user task entities that encapsulate information about the execution. The case study also aims to show examples of utilizing various GridDebug functions.

5.4.1 Software Execution

ISENGARD provides two different ways to execute grid applications: through an IDE (Eclipse RDT) or through scripts. Eclipse RDT provides remote launching support for multiple remote resources that is integrated into the IDE, as described in Section 3.5.2.2. The second method is to call Worqbench services directly through scripts. This approach allows greater flexibility and manipulation of the details of grid software execution, at the expense of writing scripts.

Eclipse RDT leverages Worqbench APIs and various launching support in Eclipse, such as output windows, launch history, and so forth. Figure 5.24 shows the Eclipse launch configuration dialog box that is used to specify the grid software execution whilst Figure 5.25 shows the Session Configuration dialog box to describe the configuration details of the execution.

As a demonstration, we executed the Jacobi application on all SSH resources (four machines) in the testbed according to the launch configuration shown in Figures 5.24 and 5.25. Figure 5.26 shows the result of the execution. The bottom pane in the figure is the standard Eclipse Console view that displays the application output from `newbruce`. The view has a drop-down menu to display the output from other resources.

As described in Sections 3.4.2 and 4.2.4, each Worqbench API call (Section 4.2.5) creates a user task object. The object stores information such as the time, status, output, and target destination of the operation. The tasks are created asynchronously and run in parallel across all participating target resources. The user task object provides a job abstraction and its asynchronous mechanism allows users to inspect *long-running* or *deferred* grid operations at a later time. For example, a grid application execution

5.4. CASE STUDY 3: SOFTWARE EXECUTION AND DEBUGGING151

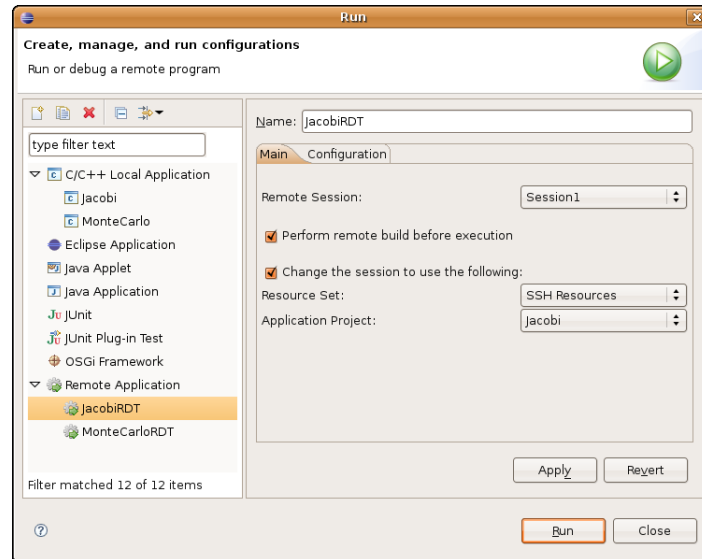


Figure 5.24: Eclipse launch configuration dialog box (run)

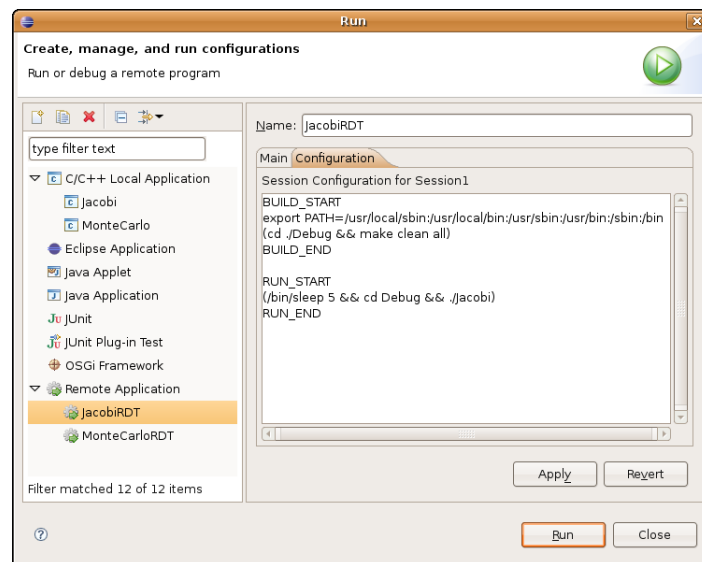


Figure 5.25: Session Configuration dialog box

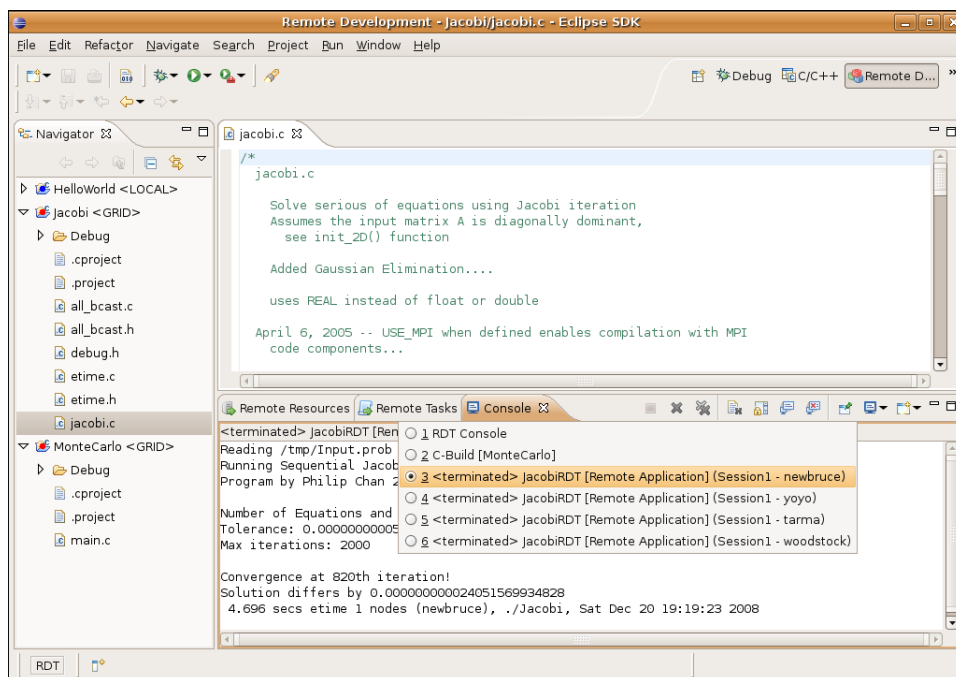


Figure 5.26: Execution of the Jacobi application

may take several days to complete or it may be scheduled at a later time by the job scheduler. The example in Figure 5.26 generated 16 user task objects (four tasks for each resource to transfer, initialize, and build the Jacobi project; and run the application). These tasks can be inspected from the Remote Tasks view (Figure 5.27) or through the Worqbench web portal page (Figure 5.28). The web interface is especially beneficial. For example, it allows users to launch grid applications from Eclipse RDT, close the IDE, login to the Worqbench web portal from another workstation, and inspect the user tasks.

The second method for running grid applications is by calling Worqbench API methods directly from scripts. This approach offers a powerful and flexible mechanism to conduct grid software execution. In spite of the fact that it involves low-level operations and details as shown in the next two figures (Figures 5.29 and 5.30), users can customize and manipulate grid software execution according to their requirements and specifications.

To demonstrate the software execution, Figures 5.29 and 5.30 show a

5.4. CASE STUDY 3: SOFTWARE EXECUTION AND DEBUGGING153

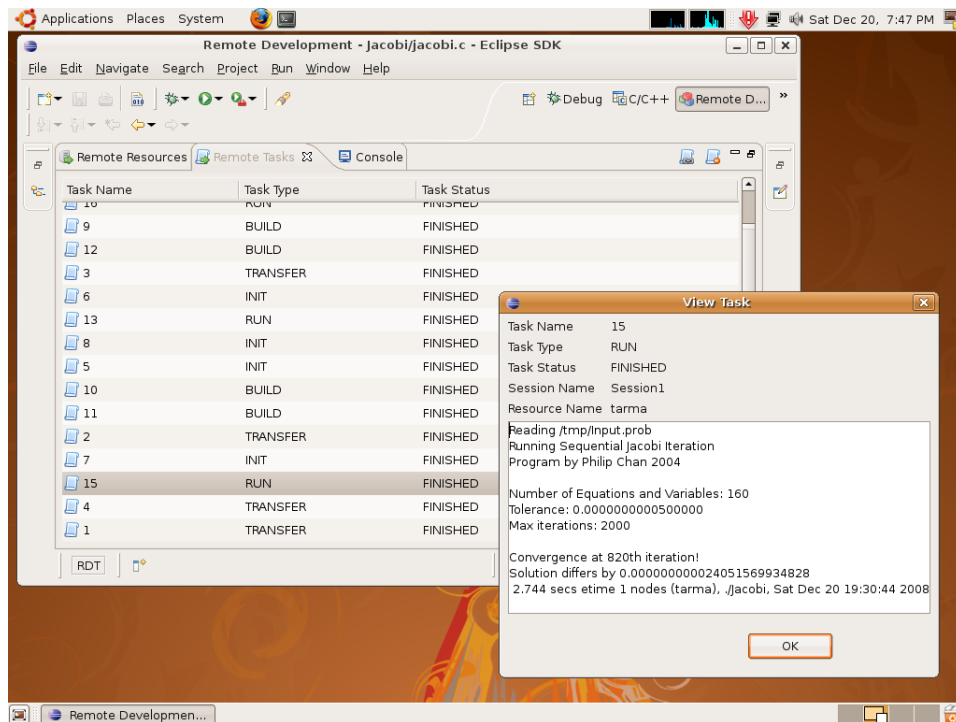


Figure 5.27: Remote Tasks view

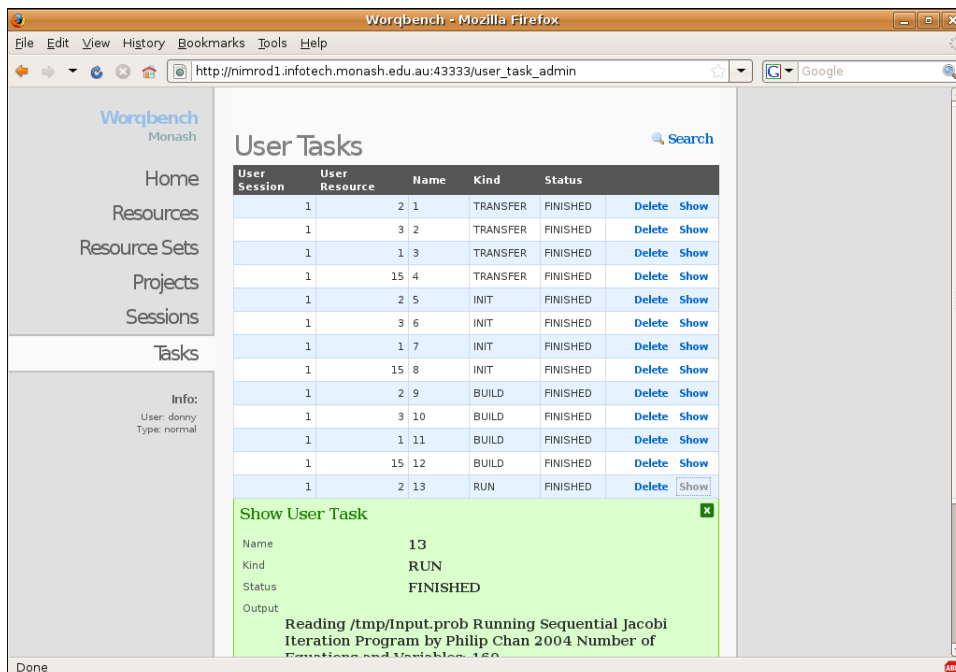


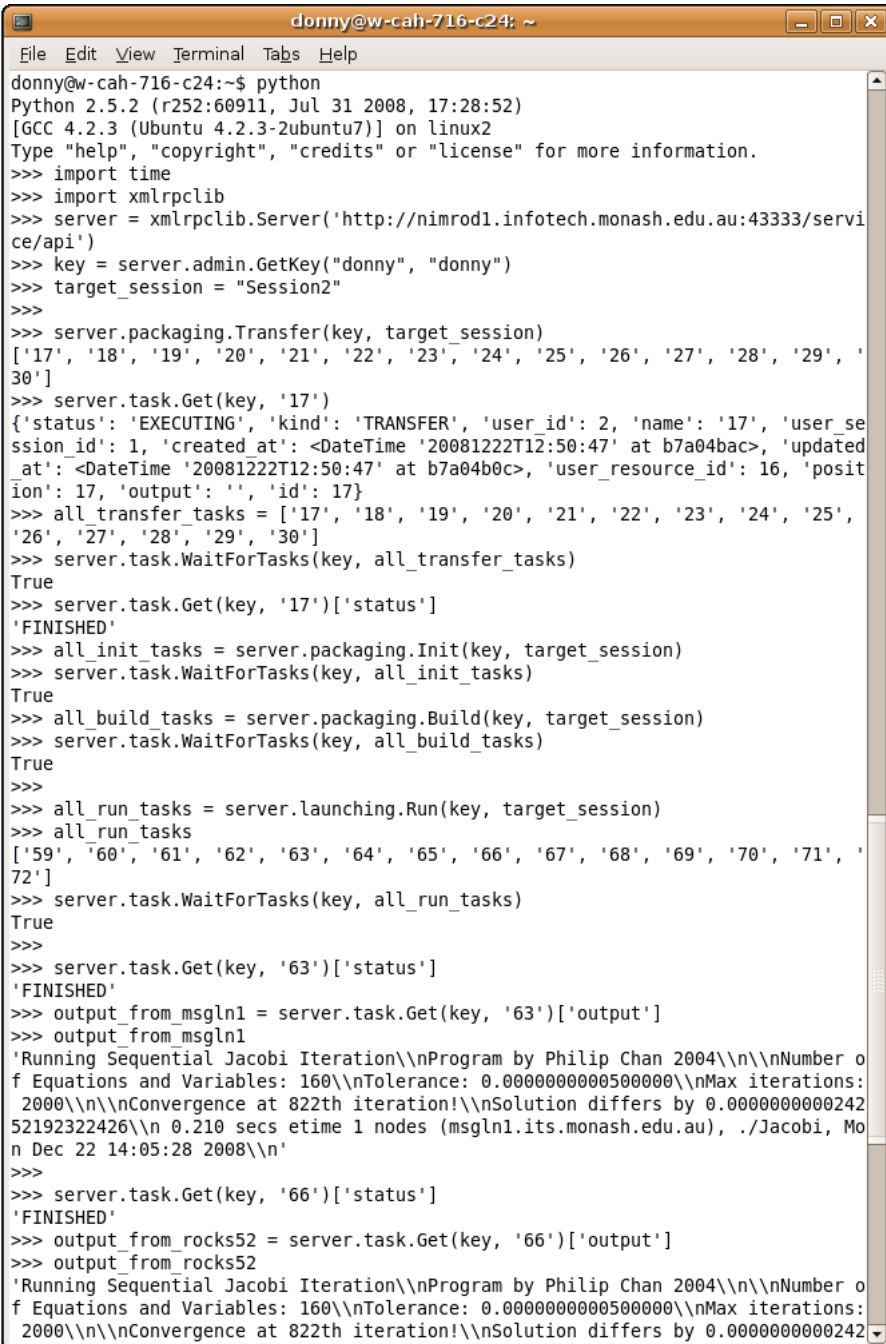
Figure 5.28: Management page for user tasks

single Python session that utilizes Worqbench services to run the Jacobi application on all resources (14 machines) in the testbed. The Python session is described as follows:

- The connection to Worqbench is initialized at lines 7-9.
- At line 10, the name of a target session is stored into a variable. "Session2" is a predefined session that is associated with the "Jacobi" project and all resources in the testbed.
- The project is copied to all resources by calling the `packaging.Transfer()` method (lines 12-14). It is a non-blocking method and returns an array of 14 tasks. The first array member (task 17) corresponds to the first resource (`tarma`) in Table 5.1 whilst the last array member (task 30) corresponds to the last resource (`alces`). At line 22, the blocking method of `task.WaitForTasks()` is called to wait for all tasks to finish.
- The project is then compiled to produce the Jacobi application. It is initialized and built by calling the `packaging.Init()` (lines 26-28) and `packaging.Build()` (lines 29-31) methods respectively.
- The application is executed on all resources by calling the `launching.Run()` method at line 33. Lines 40-95 show the status of executions and outputs on various resources. In particular, on `msgln1`, a GT4-SGE resource (lines 40-48); on `rocks-52`, a GT2-SGE resource (lines 50-62); on `sakura`, a GT4 resource (lines 65-73); on `pragma001`, a GT2 resource (lines 76-84); and on `alces`, a UNICORE resource (lines 87-95).
- The project is then deleted on remote resources by calling the `packaging.Clean()` method (lines 98-104).

The ISENGARD infrastructure supports two different methods of grid software executions that allow ISENGARD to accommodate a variety of user requirements and scenarios. It offers a familiar IDE interface to launch grid applications on remote resources; and API methods that can be utilized through scripts for specialized grid software execution.

5.4. CASE STUDY 3: SOFTWARE EXECUTION AND DEBUGGING 155



```

1 donny@w-cah-716-c24:~$ python
2 Python 2.5.2 (r252:60911, Jul 31 2008, 17:28:52)
3 [GCC 4.2.3 (Ubuntu 4.2.3-2ubuntu7)] on linux2
4 Type "help", "copyright", "credits" or "license" for more information.
5 >>> import time
6 >>> import xmlrpclib
7 >>> server = xmlrpclib.Server('http://nimrod1.infotech.monash.edu.au:43333/service/api')
8 >>> key = server.admin.GetKey("donny", "donny")
9 >>> target_session = "Session2"
10 >>>
11 >>> server.packaging.Transfer(key, target_session)
12 ['17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30']
13 >>> server.task.Get(key, '17')
14 {'status': 'EXECUTING', 'kind': 'TRANSFER', 'user_id': 2, 'name': '17', 'user_session_id': 1, 'created at': <DateTime '20081222T12:50:47' at b7a04bac>, 'updated at': <DateTime '20081222T12:50:47' at b7a04b0c>, 'user_resource_id': 16, 'position': 17, 'output': '', 'id': 17}
15 >>> all_transfer_tasks = ['17', '18', '19', '20', '21', '22', '23', '24', '25', '26', '27', '28', '29', '30']
16 >>> server.task.WaitForTasks(key, all_transfer_tasks)
17 True
18 >>> server.task.Get(key, '17')['status']
19 'FINISHED'
20 >>> all_init_tasks = server.packaging.Init(key, target_session)
21 >>> server.task.WaitForTasks(key, all_init_tasks)
22 True
23 >>> all_build_tasks = server.packaging.Build(key, target_session)
24 >>> server.task.WaitForTasks(key, all_build_tasks)
25 True
26 >>>
27 >>> all_run_tasks = server.launching.Run(key, target_session)
28 >>> all_run_tasks
29 ['59', '60', '61', '62', '63', '64', '65', '66', '67', '68', '69', '70', '71', '72']
30 >>> server.task.WaitForTasks(key, all_run_tasks)
31 True
32 >>>
33 >>> server.task.Get(key, '63')['status']
34 'FINISHED'
35 >>> output_from_msgln1 = server.task.Get(key, '63')['output']
36 >>> output_from_msgln1
37 'Running Sequential Jacobi Iteration\\nProgram by Philip Chan 2004\\n\\nNumber of Equations and Variables: 160\\nTolerance: 0.000000000500000\\nMax iterations: 2000\\n\\nConvergence at 822th iteration!\\nSolution differs by 0.00000000024252192322426\\n 0.210 secs etime 1 nodes (msgln1.its.monash.edu.au), ./Jacobi, Mon Dec 22 14:05:28 2008\\n'
38 >>>
39 >>> server.task.Get(key, '66')['status']
40 'FINISHED'
41 >>> output_from_rocks52 = server.task.Get(key, '66')['output']
42 >>> output_from_rocks52
43 'Running Sequential Jacobi Iteration\\nProgram by Philip Chan 2004\\n\\nNumber of Equations and Variables: 160\\nTolerance: 0.000000000500000\\nMax iterations: 2000\\n\\nConvergence at 822th iteration!\\nSolution differs by 0.00000000024252192322426\\n 0.210 secs etime 1 nodes (msgln1.its.monash.edu.au), ./Jacobi, Mon Dec 22 14:05:28 2008\\n'

```

Figure 5.29: Using the Worqbench API to run grid software (1)

```

57 >>> output_from_rocks52
58 'Running Sequential Jacobi Iteration\\nProgram by Philip Chan 2004\\n\\nNumber o
59 f Equations and Variables: 160\\nTolerance: 0.000000000500000\\nMax iterations:
60 2000\\n\\nConvergence at 822th iteration!\\nSolution differs by 0.000000000242
61 51215725255\\n 0.705 secs etime 1 nodes (rocks-52.sdsc.edu), ./Jacobi, Sun Dec 2
62 1 19:06:04 2008\\n'
63 >>>
64 >>>
65 >>> server.task.Get(key, '67')['status']
66 'FINISHED'
67 >>> output_from_sakura = server.task.Get(key, '67')['output']
68 >>> output_from_sakura
69 'Running Sequential Jacobi Iteration\\nProgram by Philip Chan 2004\\n\\nNumber o
70 f Equations and Variables: 160\\nTolerance: 0.000000000500000\\nMax iterations:
71 2000\\n\\nConvergence at 822th iteration!\\nSolution differs by 0.000000000242
72 52192322426\\n 0.250 secs etime 1 nodes (sakura.hpcc.jp), ./Jacobi, Mon Dec 22 1
73 2:02:09 2008\\n'
74 >>>
75 >>>
76 >>> server.task.Get(key, '69')['status']
77 'FINISHED'
78 >>> output_from_pragma001 = server.task.Get(key, '69')['output']
79 >>> output_from_pragma001
80 'Running Sequential Jacobi Iteration\\nProgram by Philip Chan 2004\\n\\nNumber o
81 f Equations and Variables: 160\\nTolerance: 0.000000000500000\\nMax iterations:
82 2000\\n\\nConvergence at 822th iteration!\\nSolution differs by 0.000000000242
83 51215725255\\n 0.625 secs etime 1 nodes (pragma001.grid.sinica.edu.tw), ./Jacobi
84 , Mon Dec 22 03:03:46 2008\\n'
85 >>>
86 >>>
87 >>> server.task.Get(key, '72')['status']
88 'FINISHED'
89 >>> output_from_alces = server.task.Get(key, '72')['output']
90 >>> output_from_alces
91 'Running Sequential Jacobi Iteration\\nProgram by Philip Chan 2004\\n\\nNumber o
92 f Equations and Variables: 160\\nTolerance: 0.000000000500000\\nMax iterations:
93 2000\\n\\nConvergence at 822th iteration!\\nSolution differs by 0.000000000242
94 52192322426\\n 0.210 secs etime 1 nodes (alces), ./Jacobi, Mon Dec 22 04:07:18 2
95 008\\n'
96 >>>
97 >>>
98 >>> server.packaging.Clean(key, target_session)
99 ['73', '74', '75', '76', '77', '78', '79', '80', '81', '82', '83', '84', '85', '
100 86']
101 >>> all_clean_tasks = ['73', '74', '75', '76', '77', '78', '79', '80', '81', '82
102 ', '83', '84', '85', '86']
103 >>> server.task.WaitForTasks(key, all_clean_tasks)
104 True
105 >>>
106 donny@w-cah-716-c24:~$

```

Figure 5.30: Using the Worqbench API to run grid software (2)

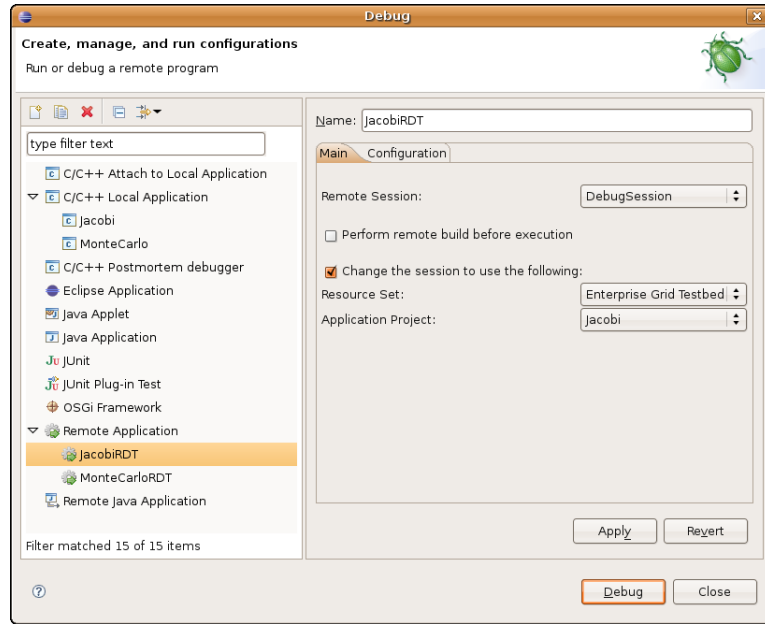


Figure 5.31: Eclipse launch configuration dialog box (debug)

5.4.2 Debugging Grid Applications

Similar to the software execution, debugging grid applications can be performed by using Eclipse RDT or by calling GridDebug API methods directly (Sections 3.3.2 and 4.1.2).

Eclipse RDT provides a simple, yet advanced grid debugging environment, by leveraging various debugging support in the IDE, such as a variable view, a breakpoint view, a stack trace view, and so forth. The Eclipse launch configuration dialog box as shown in Figure 5.31 is used to launch a grid application under the control of a debugger.

As a demonstration, we debugged the Jacobi application on two worker nodes in the Enterprise Grid. As introduced in Section 5.1, the grid infrastructure consists of two compute clusters, `east-lin` and `west-lin`. Each cluster employs the GT4 grid middleware, the GridDebug service, and the Sun Grid Engine job scheduler. The Jacobi application is submitted as a GT4-SGE job on each cluster.

The debugging exercise is shown in Figures 5.32 and 5.33. The figures display the Eclipse Debug perspective that organizes views and editors re-

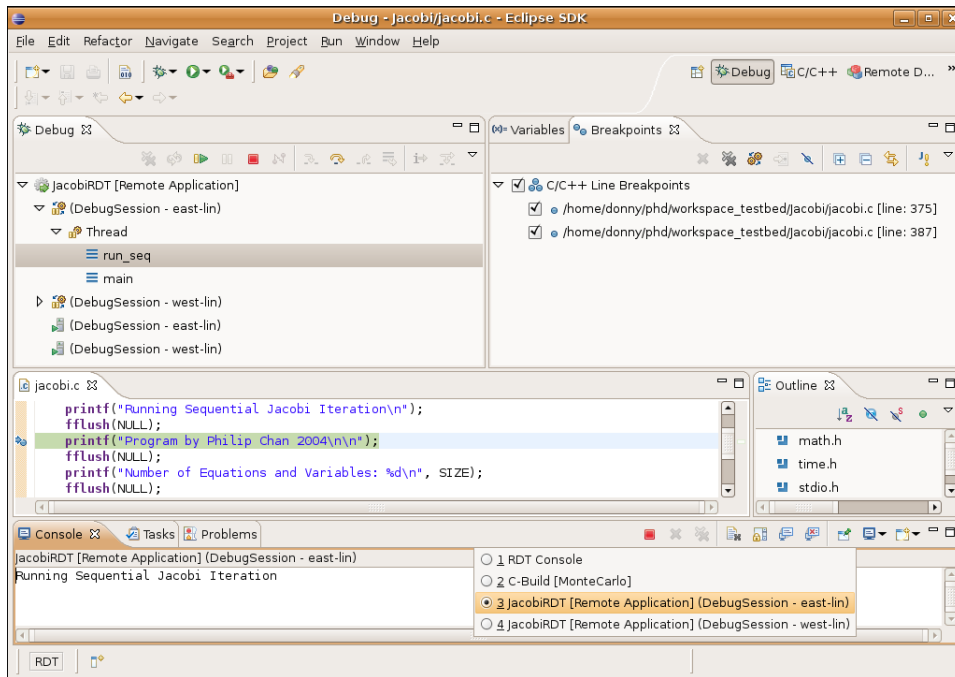


Figure 5.32: Eclipse Debug perspective (1)

lated to software debugging. In both figures, the top left, centre left, centre right, and bottom panes display the Eclipse Debug view, code editor, Outline view, and Console view respectively. The Debug view displays stack traces and processes being debugged, whilst the editor highlights the current line in the file `jacobi.c`. Breakpoints can be specified through the Breakpoints view (the top right pane in Figure 5.32) whilst the Variables view (the top right pane in Figure 5.33) shows valid variables in the current stack frame.

Figure 5.34 shows the topology of the resources used in this debugging exercise. As described in Section 3.3.3, the GridDebug service is notified when the application execution begins. The GridDebug service then notifies the Worqbench Launching Service, which in turn notifies Eclipse RDT. The ISENGARD software infrastructure (GridDebug, Worqbench, and Eclipse RDT) transparently handles control and data communication between participating resources. ISENGARD eliminates the need for ad-hoc debugging techniques, such as polling the scheduler regularly to check whether the application has started and manually attaching a debugger to the process. Fur-

5.4. CASE STUDY 3: SOFTWARE EXECUTION AND DEBUGGING159

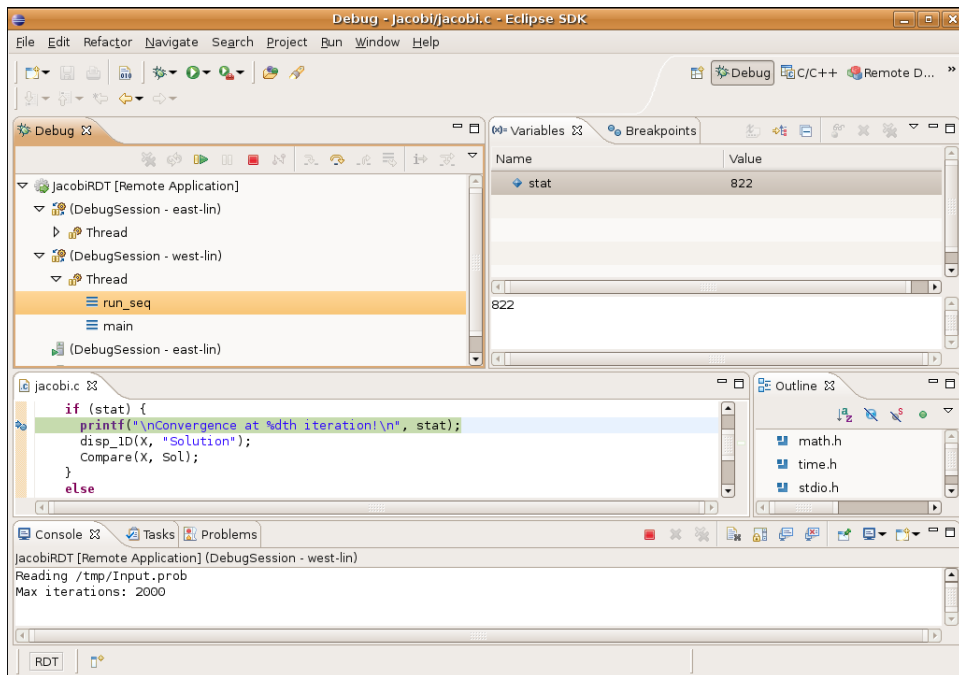


Figure 5.33: Eclipse Debug perspective (2)

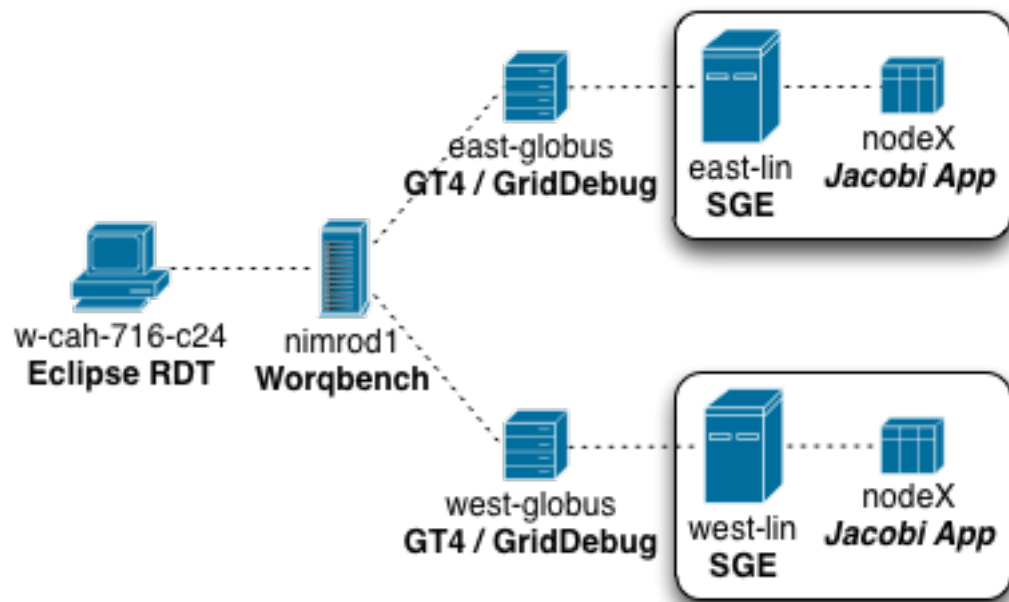


Figure 5.34: Topology of the debugging demonstration

thermore, it allows the debugging activity to be performed from a simple yet advanced grid development environment.

In addition to utilizing the IDE to debug grid applications, it is also possible to write debug clients that employ and call the GridDebug API methods directly. It allows developers to write sophisticated debug clients, for example, a high-level tracer or a grid-enabled relative debugger [Abramson and Sasic, 1995, Watson, 2000]. Relative debugging is a technique that enables a programmer to locate and identify errors by comparing data structures in a suspect program against a reference code. Appendix D lists two examples of GridDebug debug clients.

Listing D.1 in Appendix D shows a command-line interface grid debugger that employs the GridDebug service. The program, a very short Java code, accepts user input and calls the appropriate API methods. It subscribes to a notification service to get program and debugger outputs. This simple yet fully functional debug client demonstrates the use of GridDebug and its API.

Listing D.2 in Appendix D shows another example of GridDebug debug clients. It is a Jython [Jython, 2008] script that compares variables from two Jacobi applications running on `east-globus` and `west-globus`. It performs a comparison between data variables which is the key idea of relative debugging.

This section has demonstrated the debugging functions provided by the ISENGARD software infrastructure. It supports grid software debugging from IDEs or custom debug clients. ISENGARD presents workable solutions to the grid debugging problems identified in Chapter 1.

5.5 Case Study 4: Molecular Dynamics Simulation

The main objective of the fourth case study is to demonstrate the applicability of ISENGARD on typical large-scale scientific projects. This case study concerns a substantial software package for molecular dynamics simulation, called GROMACS [GROMACS, 2009]. The software package consists

of 300,000 lines of C code across 700 source files. Specifically, the case study demonstrates: preparing the Eclipse project for GROMACS and the testbed resources; editing GROMACS source files with the multi-view code editor; compiling the software package by utilizing the API methods; and launching the test application that is provided in the GROMACS distribution. In addition, to show the variety of user interactions, the development activities in this case study are conducted through various access methods: the web portal, the user scripts, and the development environment. The case study, however, does not repeat details that have been presented in the previous sections. The case study is conducted as follows:

1. We downloaded the source code bundle of GROMACS version 3.3.3 and imported it as a C software project (**Gromacs**) into Eclipse RDT. Then, we built the project and executed the test application locally as displayed in Figure 5.35. This procedure was conducted with ease due to the maturity of Eclipse; and as shown in Figure 5.35 the IDE can handle a large-scale software package like GROMACS.
2. We set up a testbed set (**Gromacs Testbed**) that consists of three resources through the Worqbench web portal. The resources are: **eastlin** (GT4-SGE), **newbruce** (SSH), and **rocks-52** (GT2-SGE). The resource details can be found in Table 5.1 in Section 5.1. We then created a new remote project (**Gromacs**) and a new development session (**Gromacs Session**) as shown in Figure 5.36.
3. As introduced in Section 5.1, GROMACS is a multi-platform software package with specialized low-level instruction sets for different processors. It allows GROMACS to perform fast computations by taking advantage of the processors' distinct capabilities.

The low-level code variations are written by employing pre-processor and macro directives that may not be easy to understand. For example, the left pane in Figure 5.37 shows one of the platform related C files opened with the standard Eclipse C/C++ editor. We used the multi-view code editor to simplify code comprehension. The right pane

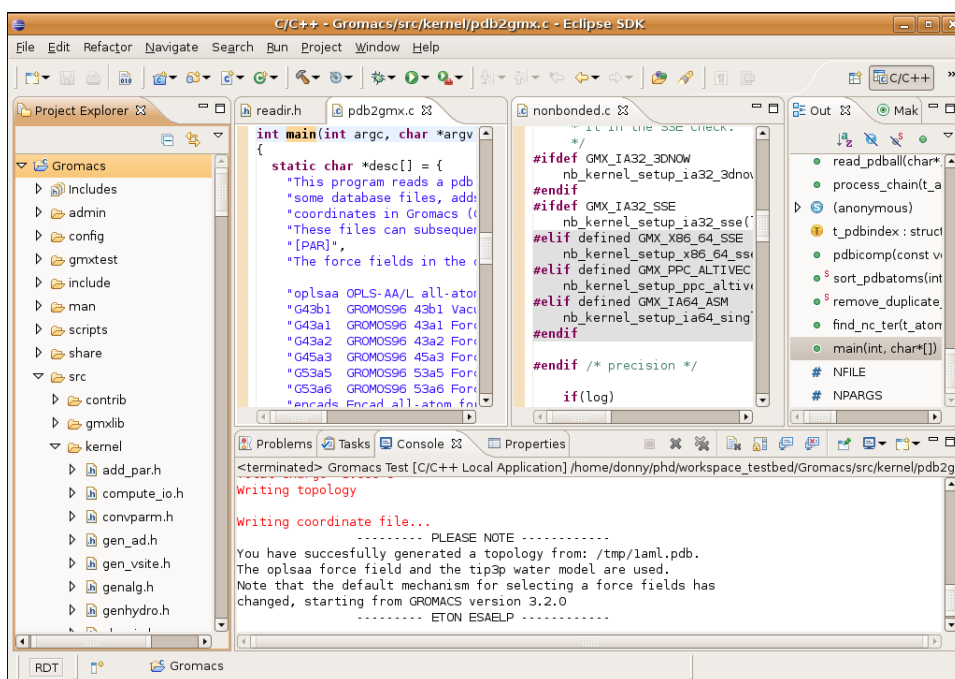


Figure 5.35: Developing GROMACS in Eclipse

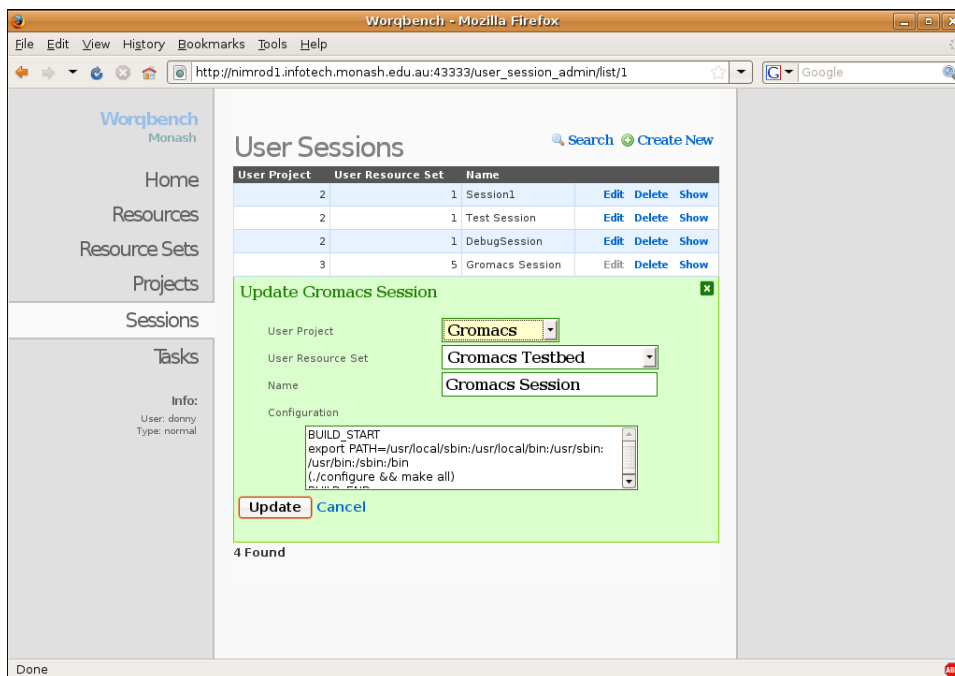


Figure 5.36: GROMACS development session

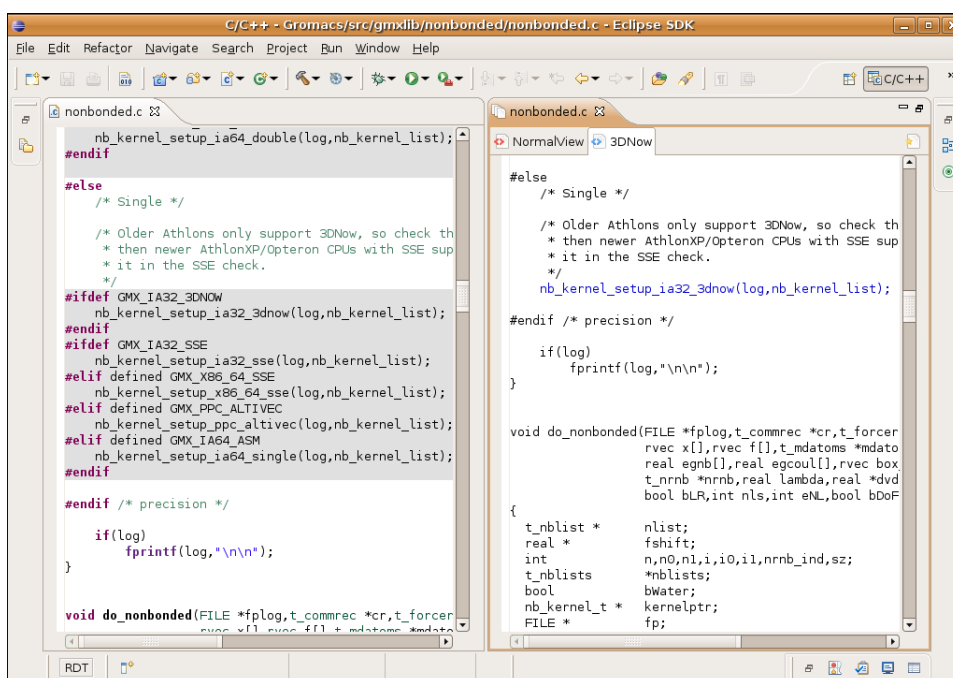
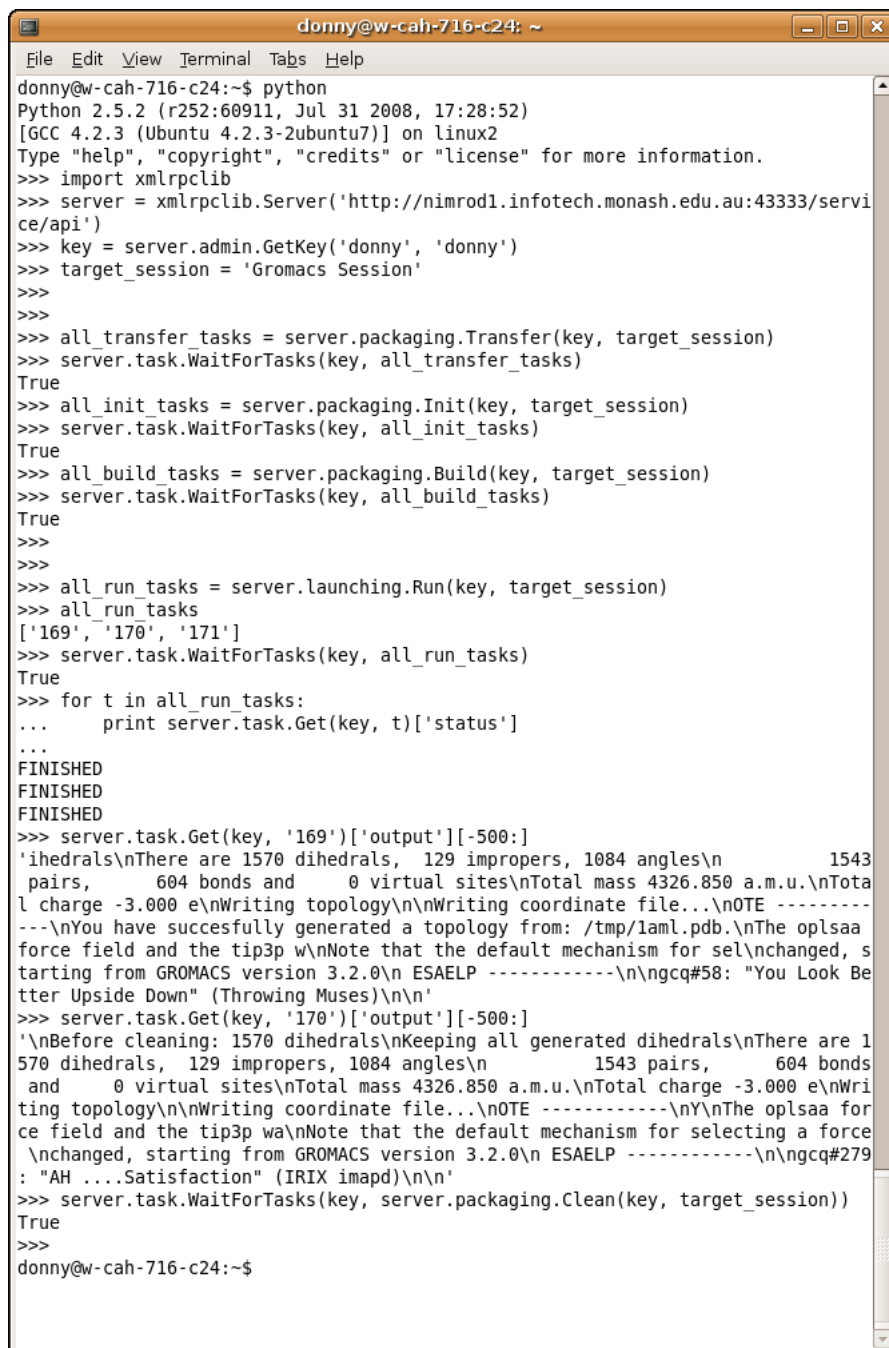


Figure 5.37: GROMACS source code in two different editors

in Figure 5.37 shows the same file opened with the multi-view editor. It displays the source code related to the 3DNow multimedia extension (GMX_IA32_3DNow). As described in Section 3.5.2.3, the multi-view editor helps developers by showing code portions that are relevant in a certain context and hiding the irrelevant parts.

4. We uploaded the local GROMACS project in Eclipse RDT into the Worqbench remote project. The GROMACS project was then built and its test application was executed on remote resources by calling Worqbench service APIs as demonstrated in Figure 5.38. The figure also shows the result of the successful executions on the testbed (the output was trimmed due to its large size).

This case study demonstrates that the ISENGARD software infrastructure can be used with a large-scale project such as GROMACS. It helped us with various, otherwise laborious operations, such as manually copying the files to multiple resources and logging into the nodes to execute the applica-



```

donny@w-cah-716-c24:~$ python
Python 2.5.2 (r252:60911, Jul 31 2008, 17:28:52)
[GCC 4.2.3 (Ubuntu 4.2.3-2ubuntu7)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import xmlrpclib
>>> server = xmlrpclib.Server('http://nimrod1.infotech.monash.edu.au:43333/service/api')
>>> key = server.admin.GetKey('donny', 'donny')
>>> target_session = 'Gromacs Session'
>>>
>>>
>>> all_transfer_tasks = server.packaging.Transfer(key, target_session)
>>> server.task.WaitForTasks(key, all_transfer_tasks)
True
>>> all_init_tasks = server.packaging.Init(key, target_session)
>>> server.task.WaitForTasks(key, all_init_tasks)
True
>>> all_build_tasks = server.packaging.Build(key, target_session)
>>> server.task.WaitForTasks(key, all_build_tasks)
True
>>>
>>>
>>> all_run_tasks = server.launching.Run(key, target_session)
>>> all_run_tasks
['169', '170', '171']
>>> server.task.WaitForTasks(key, all_run_tasks)
True
>>> for t in all_run_tasks:
...     print server.task.Get(key, t)['status']
...
FINISHED
FINISHED
FINISHED
>>> server.task.Get(key, '169')['output'][-500:]
'ihedrals\nThere are 1570 dihedrals, 129 impropers, 1084 angles\n          1543
pairs,          604 bonds and          0 virtual sites\nTotal mass 4326.850 a.m.u.\nTotal
charge -3.000 e\nWriting topology\n\nWriting coordinate file...\nNOTE -----
---\nYou have succesfully generated a topology from: /tmp/1aml.pdb.\nThe oplsa
force field and the tip3p wa\nNote that the default mechanism for sel\nchanged, s
tarting from GROMACS version 3.2.0\n ESAELP ----- \n\ngcq#58: "You Look Be
tter Upside Down" (Throwing Muses)\n\n'
>>> server.task.Get(key, '170')['output'][-500:]
'\nBefore cleaning: 1570 dihedrals\nKeeping all generated dihedrals\nThere are 1
570 dihedrals, 129 impropers, 1084 angles\n          1543 pairs,          604 bonds
and          0 virtual sites\nTotal mass 4326.850 a.m.u.\nTotal charge -3.000 e\nWri
ting topology\n\nWriting coordinate file...\nNOTE ----- \nY\nThe oplsa
force field and the tip3p wa\nNote that the default mechanism for selecting a force
\nchanged, starting from GROMACS version 3.2.0\n ESAELP ----- \n\ngcq#279
: "AH ...Satisfaction" (IRIX imapd)\n\n'
>>> server.task.WaitForTasks(key, server.packaging.Clean(key, target_session))
True
>>>
donny@w-cah-716-c24:~$

```

Figure 5.38: Testing the GROMACS package

tion.

5.6 System Demonstration Summary

This chapter has presented a system demonstration of the ISENGARD software infrastructure. It described a series of case studies and showed the use and advantage of GridDebug, Worqbench, and RDT in the grid software development process.

The first case study (Section 5.2) demonstrates a broad range of options to access ISENGARD services, for example, through a standard web browser, an IDE, or user scripts. In addition, we have also demonstrated management functions, such as, initializing access to grid resources, setting up resource sets, and managing application projects and sessions.

The second case study (Section 5.3) demonstrates the use of IDE scripting and the multi-view code editor. We illustrated the IDE automation by listing a number of Groovy scripts that can perform various activities on behalf of the developers. Additionally, we have shown the process of editing a source file with the multi-view code editor. The editor shows only relevant portions of the code and hides the irrelevant parts. It simplifies code comprehension of multi-platform software.

The third case study (Section 5.4) demonstrates grid software execution and debugging performed by utilizing GridDebug, Worqbench, and RDT. We have shown that it is possible to use sophisticated tools such as IDEs to execute and debug remote grid software. In addition, we have demonstrated that the execution and debugging services provided by ISENGARD can be extended or customized to perform specialized development tasks, for example, grid relative debugging.

The fourth case study (Section 5.5) demonstrates the applicability of ISENGARD on a large molecular dynamics simulation package called GROMACS. We have shown this by performing a range of operations such as editing GROMACS with the multi-view code editor; compiling the software package; and launching the test application on multiple resources.

In conclusion, by presenting and conducting a series of four case stud-

ies, we can demonstrate that the overall ISENGARD software infrastructure works and that the proof-of-concept implementations provide workable solutions to the problems and challenges identified in Chapter 1.

Chapter 6

Future Directions and Conclusions

Grid computing facilitates the aggregation of resources that are distributed across multiple administrative domains to act as a single large system. The paradigm allows e-Scientists to conduct scientific research with increasing scale, complexity, and scope. To take advantage of the grid resources, applications need to be specifically written with careful attention to the characteristic of the infrastructure. The process of writing, deploying, and testing grid applications is complex and challenging. To date, no one has presented a complete solution of grid-enabled programming tools that can support e-Scientists in developing these applications effectively and efficiently. The lack of such tools has encumbered e-Scientists from collaborating and utilizing grid infrastructure for large-scale scientific research.

As a solution, we proposed ISENGARD, a software infrastructure for supporting e-Science and grid application development. Specifically, we presented the design, implementation, and demonstration of ISENGARD which consists of three software components that solve key issues in distributed debugging, grid development services, and grid IDEs: GridDebug, Worqbench, and Remote Development Tools (RDT).

This chapter presents a summary of significant research contributions of the thesis (Section 6.1), future research directions in the area of grid appli-

cation development (Section 6.2), and the thesis conclusions (Section 6.3).

6.1 Thesis Summary

In this thesis, we have addressed important issues in grid application development. Specifically, we have:

- Identified the challenges and issues in the area of grid application development (Section 1.1).
- Presented the background and related work (Chapter 2). We have discussed and assessed that no existing work provides a comprehensive solution and addresses the aforementioned problems.
- Identified the design objectives for a grid development environment (Section 3.1). On the basis of these objectives, we have designed and presented a new infrastructure that supports the development of e-Science and grid applications (ISENGARD) (Section 3.2). It employs a modular and layered software architecture that allows loose coupling between various components.
- Designed and developed a grid service debug architecture, GridDebug (Sections 3.3 and 4.1), that can be incorporated into any existing grid middleware for testing and debugging grid applications.
- Presented the specification and design of an application programming interface for grid debugging (Sections 3.3.2 and 4.1.2).
- Designed and developed a modular and extensible framework, Worqbench (Sections 3.4 and 4.2), that manages the interaction between grid middleware and IDEs. The framework provides APIs and services that correspond to the stages in the implementation phase of a software development life cycle.
- Presented the specification of a comprehensive resource and development model for computational grid applications (Section 3.4.2). It

formalizes entities such as application projects, grid resources, resource sets, and development sessions.

- Designed and developed a framework and a set of tools for IDEs, RDT (Sections 3.5 and 4.3), that promotes a tight integration between the software development process and the execution platforms. The framework offers a set of API methods and a runtime system that allow developers to write automation and event callback scripts.
- Presented the design and implementation of a multi-view code editor (Sections 3.5.2.3 and 4.3.5) that separates the display of source code from the textual representation.
- Demonstrated the ISENGARD software infrastructure using a series of four case studies that reflect common situations faced by grid software developers (Chapter 5).

6.2 Future Research Directions

This thesis has contributed to improved understanding in the area of grid application development. Even though it solves some critical questions, it has also revealed new questions and challenges that need to be addressed through a wide range of potential research avenues. Specifically, we list three future research areas: advanced application development support in grid middleware (Section 6.2.1), innovative grid development environments (Section 6.2.2), and development tools for high-level grid applications (Section 6.2.3).

6.2.1 Advanced Development Support in Grid Middleware

The GridDebug framework has been designed as a novel middleware-level tool for debugging grid applications. Nevertheless, the framework could be further extended, for example, by incorporating relative debugging functionality [Abramson and Sosic, 1995, Watson, 2000]. Relative debugging enables

a programmer to locate and identify errors by comparing data structures in a suspect program against a reference code. Thus, it is possible to utilise the information contained in prior, but correctly functioning software, to support the process of debugging a new version of the program. A grid-based relative debugger would be beneficial for e-Scientists porting legacy programs to grid applications.

Worqbench is an upper middleware software framework that provides high-level grid development services and API. It could be extended to provide additional services such as performance profiling and automated unit testing. These services allow improved software analysis that could lead to robust and bug-free grid applications.

6.2.2 Innovative Grid Development Environments

RDT has advanced the state of grid IDEs with its features such as an IDE scripting facility, a multi-view code editor, and an event mechanism with user-defined callback methods. RDT has laid the groundwork for further enhancements and innovative research in the area of grid IDEs. For example: a tool for intelligent matchmaking of software projects with suitable target platforms; an extensive scripting facility to automate application deployment or debugging; and editing tools that facilitate real-time collaboration between e-Scientists. The editors could leverage social networking sites such as Facebook [Facebook, 2009] or grid-specific ones such as myExperiment [myExperiment, 2009]. These new tools could potentially make the task of developing grid applications easier and more efficient.

Furthermore, with the advent of multi-core processors and hardware virtualization, considerable research could be devoted to design and implement highly scalable grid development environments. These grid IDEs need to be able to handle thousands of processes and resources, which could benefit e-Science research with greater complexity and scale.

6.2.3 Development Tools for High-Level Grid Applications

ISENGARD has been designed and implemented for traditional grid applications in the form of program executables. Nevertheless, we foresee that ISENGARD could be extended to support high-level grid software such as grid services, workflows, and parameter sweep applications. There are, however, open questions that need to be addressed. For example: "How do we debug the interaction of software components in a grid workflow?" or "What is the best way to profile a parameter sweep application?".

An implementation of ISENGARD for high-level grid applications would provide an attractive environment and tools to write, deploy, and debug the applications in an efficient manner.

6.3 Thesis Conclusions

To conclude, we have designed, developed, and demonstrated an innovative software infrastructure for supporting e-Science and grid application development. The infrastructure consists of a modular collection of "pluggable" grid development tools as opposed to a single monolithic system. We identify and present several significant outcomes and conclusions in this thesis:

- Increasing the scalability of the development process.

ISENGARD provides high-level abstractions and services that can perform development tasks on behalf of programmers across a number of grid resources, rather than requiring the users to perform the tasks manually for each individual machine.

GridDebug is designed as component-based software rather than a stand-alone monolithic program; and it utilizes an agent-based callback mechanism for grid remote debugging. This design allows sophisticated debug clients to be written that handle grid debugging on multiple remote resources. It helps programmers avoid ad-hoc techniques such as polling the scheduler regularly to check whether the application has

started and manually attaching a debugger to the process. GridDebug allows programmers to perform grid debugging in effective and efficient manners.

The Worqbench system model formalizes the entities and helps the users to manage and utilize them. In addition, the client-server design of Worqbench allows it to perform "offline" or unattended operations with the projects on users' behalf. For example, executing automatic builds or test harnesses on distributed resources.

RDT offers a runtime programmable facility that allows the IDE to be scripted during its execution. RDT scripts can be coded to automate and repeat many tedious grid programming tasks, such as software compilation and unit testing, across a variety of resources. Thus, this automation simplifies grid application development for a large number of target platforms efficiently, a process which is significantly more complex than when software is developed for a single platform. Moreover, scripting allows the user to tailor these functions for variations in particular grid infrastructure.

- Addressing the issues of heterogeneity.

The ISENGARD infrastructure provides adapters, connectors, and plug-ins that can operate with a variety of grid tools and systems. More importantly, its versatile design allows the infrastructure to support future grid software.

GridDebug provides a set of API methods that abstracts the debug back-ends from the application developers and it presents uniform access to debugging and testing functions.

Similarly, Worqbench provides application development services for building, testing, managing, and deploying grid-enabled software on resources managed by different grid middleware stacks. Worqbench presents a unified view of the underlying services to the developers.

RDT with its framework and plug-in system enables traditional IDEs to be used for grid application development. It doesn't necessitate a

particular development environment to be used.

- Proposing a solution to the lack of standards.

ISENGARD components such as GridDebug, Worqbench, and RDT, provide API methods and services that cover a wide range of programming functions. These functions correspond to the stages in the implementation phase of a software development life cycle. We believe, the APIs could be adopted as open standards of grid development services that can work with various tools and libraries.

In addition, ISENGARD components are modular and extensible; and they can accommodate various IDEs, grid middleware stacks, and debug back-ends. Thus, this design increases the level of compatibility and interoperability between ISENGARD and other grid tools and systems.

- Providing user-friendly yet powerful programming tools.

RDT extends traditional IDEs to work with remote and distributed grid resources. It lowers the barrier to entry for programmers accustomed to traditional application development by providing a familiar and intuitive grid programming environment. The RDT multi-view code editor also simplifies code comprehension for programmers developing software for multiple architectures.

Alternatively, Worqbench provides a client connector that allows high-level functions to be accessed through a web browser without the need to install specialized software. The web portal enables users and e-Scientists to easily access grid resources for program execution and aggregate experiment results and information into a single web interface.

GridDebug simplifies grid application debugging by handling complex issues such as job scheduling and access restrictions across a hierarchy of resources.

In addition to being user-friendly and easy to use, ISENGARD components also come with a rich collection of API and access methods that

provide flexible and powerful mechanisms to utilize the grid development services.

Appendix A

GridDebug API Methods

```
public abstract void load(String progFilePath);
public abstract void load(String progFilePath, int numProcs);
public abstract void run();
public abstract void run(String[] progArgs);
public abstract void exit();
```

Table A.1: Debugger initialization and termination methods

```
public abstract void focus(String setName);
public abstract void defSet(String setName, int[] procsIds);
public abstract void undefSet(String setName);
public abstract void undefSetAll();
public abstract DebugProcess[] viewSet(String setName);
```

Table A.2: Methods associated with process sets

```
public abstract String print(String expression);
public abstract void assign(String varName, String varValue);
public abstract DebugVariable[] listVariables();
```

Table A.3: Data display and manipulation methods

| |
|--|
| <pre> public abstract void breakpoint(String location); public abstract void breakpoint(String location, int count); public abstract void breakpoint(String location, String condition); public abstract void watchpoint(String varName); public abstract DebugActionpoint[] actions(); public abstract DebugActionpoint[] actions(int[] ids); public abstract DebugActionpoint[] actions(String type); public abstract void delete(int[] ids); public abstract void delete(String type); public abstract void disable(int[] ids); public abstract void disable(String type); public abstract void enable(int[] ids); public abstract void enable(String type); </pre> |
| <pre> public abstract void breakpointSet(String setName, ↵ String location); public abstract void breakpointSet(String setName, ↵ String location, int count); public abstract void breakpointSet(String setName, ↵ String location, String condition); public abstract void watchpointSet(String setName, ↵ String varName); </pre> |

Table A.4: Methods associated with actionpoints

| |
|--|
| <pre> public abstract void step(); public abstract void step(int count); public abstract void stepOver(); public abstract void stepOver(int count); public abstract void stepFinish(); public abstract void halt(); public abstract void cont(); </pre> |
| <pre> public abstract void stepSet(String setName); public abstract void stepSet(String setName, int count); public abstract void stepOverSet(String setName); public abstract void stepOverSet(String setName, int count); public abstract void stepFinishSet(String setName); public abstract void haltSet(String setName); public abstract void contSet(String setName); </pre> |

Table A.5: Execution control methods

| |
|---|
| <pre> public abstract DebugStackFrame[] where(); public abstract void up(); public abstract void up(int count); public abstract void down(); public abstract void down(int count); </pre> |
|---|

Table A.6: Methods associated with program information

Appendix B

Worqbench API Methods

| Standard attributes (Resource) | Description |
|--------------------------------|--|
| id | Identification number of the resource. |
| name | Name of the resource. |
| kind | Type of the resource. |
| address | Network address of the resource. |
| port | Port number of the resource. |
| configuration | Configuration of the resource. |
| Standard attributes (User) | Description |
| id | Identification number of the user. |
| name | Name of the user. |
| kind | Type of the user. |
| password | Login password to access Worqbench. |
| Method | Description |
| admin.GetKey() | Get the session key for a user. |
| admin.NewResource() | Create a new system resource. |
| admin.DestroyResource() | Delete a system resource. |
| admin.GetResources() | Get all system resources. |
| admin.NewUser() | Create a new regular user. |
| admin.NewAdmin() | Create a new administrator. |
| admin.DestroyUser() | Delete a user. |
| admin.GetUsers() | Get all Worqbench users. |
| admin.ChangePassword() | Change the password for a user. |

Table B.1: System resource and user API methods

| Standard attributes | Description |
|------------------------------------|--|
| <code>id</code> | Identification number of the user resource. |
| <code>user_id</code> | Reference to the user resource's owner. |
| <code>resource_id</code> | Reference to the system resource. |
| <code>user_resource_set_ids</code> | Reference to resource sets. |
| <code>name</code> | Name of the user resource. |
| <code>user_name</code> | Login name to access the user resource. |
| <code>user_password</code> | Login password to access the user resource. |
| <code>user_certificate</code> | Login certificate to access the user resource. |
| Methods | Description |
| <code>resource.List()</code> | List all resources of a user. |
| <code>resource.Get()</code> | Get a particular user resource. |
| <code>resource.New()</code> | Create a new user resource. |
| <code>resource.Destroy()</code> | Delete a user resource. |
| <code>resource.Edit()</code> | Edit user resource's standard attributes. |
| <code>resource.SetAttr()</code> | Change user resource's custom attributes. |
| <code>resource.GetAttr()</code> | Get user resource's custom attributes. |

Table B.2: UserResource API methods

| Standard attributes | Description |
|-------------------------------------|---|
| <code>id</code> | Identification number of the project. |
| <code>user_id</code> | Reference to the project's owner. |
| <code>name</code> | Name of the project. |
| <code>data</code> | Compressed data archive of the project. |
| Methods | Description |
| <code>project.List()</code> | List all projects of a user. |
| <code>project.Get()</code> | Get a particular project. |
| <code>project.New()</code> | Create a new project. |
| <code>project.Destroy()</code> | Delete a project. |
| <code>project.Edit()</code> | Edit project's standard attributes. |
| <code>project.SetAttr()</code> | Change project's custom attributes. |
| <code>project.GetAttr()</code> | Get project's custom attributes. |
| <code>project.UploadData()</code> | Upload a project zip file. |
| <code>project.DownloadData()</code> | Download a project zip file. |

Table B.3: UserProject API methods

| Standard attributes | Description |
|-----------------------------------|--|
| <code>id</code> | Identification number of the resource set. |
| <code>user_id</code> | Reference to the resource set's owner. |
| <code>user_resource_ids</code> | Reference to member resources. |
| <code>name</code> | Name of the set. |
| Methods | Description |
| <code>set.List()</code> | List all resource sets of a user. |
| <code>set.Get()</code> | Get a particular resource set. |
| <code>set.New()</code> | Create a new resource set. |
| <code>set.Destroy()</code> | Delete a resource set. |
| <code>set.Edit()</code> | Edit resource set's standard attributes. |
| <code>set.AddResource()</code> | Add a resource to a set. |
| <code>set.RemoveResource()</code> | Remove a resource from a set. |
| <code>set.ListResource()</code> | List all resources in a set. |
| <code>set.SetAttr()</code> | Change resource set's custom attributes. |
| <code>set.GetAttr()</code> | Get resource set's custom attributes. |

Table B.4: UserResourceSet API methods

| Standard attributes | Description |
|-----------------------------------|---|
| <code>id</code> | Identification number of the session. |
| <code>user_id</code> | Reference to the session's owner. |
| <code>user_project_id</code> | Reference to the project of the session. |
| <code>user_resource_set_id</code> | Reference to the resource set of the session. |
| <code>name</code> | Name of the session. |
| <code>configuration</code> | Configuration of the session. |
| Methods | Description |
| <code>session.List()</code> | List all sessions of a user. |
| <code>session.Get()</code> | Get a particular session. |
| <code>session.New()</code> | Create a new session. |
| <code>session.Destroy()</code> | Delete a session. |
| <code>session.Edit()</code> | Edit session's standard attributes. |
| <code>session.SetAttr()</code> | Change session's custom attributes. |
| <code>session.GetAttr()</code> | Get session's custom attributes. |
| <code>session.SetConfig()</code> | Change session's configuration. |
| <code>session.GetConfig()</code> | Get session's configuration. |

Table B.5: UserSession API methods

| Standard attributes | Description |
|----------------------------------|--|
| <code>id</code> | Identification number of the task. |
| <code>user_id</code> | Reference to the task's owner. |
| <code>user_session_id</code> | Reference to the session of the task. |
| <code>user_resource_id</code> | Reference to the resource of the task. |
| <code>name</code> | Name of the task. |
| <code>kind</code> | Type of the task. |
| <code>status</code> | Status of the task. |
| <code>output</code> | Output of the task. |
| <code>created_at</code> | Creation time of the task. |
| <code>updated_at</code> | Modification time of the task. |
| Methods | Description |
| <code>task.List()</code> | List all tasks of a user. |
| <code>task.Get()</code> | Get a particular task. |
| <code>task.Destroy()</code> | Delete a task. |
| <code>task.DestroyAll()</code> | Delete all tasks of a user. |
| <code>task.GetStatus()</code> | Get a task's status. |
| <code>task.GetOutput()</code> | Get a task's output message. |
| <code>task.SendInput()</code> | Send an input message to a task. |
| <code>task.SendDebug()</code> | Send a debug message to a task. |
| <code>task.WaitFor()</code> | Wait for a task to complete. |
| <code>task.WaitForTasks()</code> | Wait for tasks to complete. |

Table B.6: UserTask API methods

Appendix C

Remote Development Tools API Methods

| IResource |
|---|
| <code>String getTags()</code> <code>void setTags(String tags)</code> <code>String getResourceName()</code> <code>Integer getResourceId()</code> <code>String getResourceType()</code> <code>String getResourceAddr()</code> <code>Integer getResourcePort()</code> <code>void setResourceName(String newName)</code> <code>void setResource(Integer newResource)</code> |
| ISet |
| <code>String getTags()</code> <code>void setTags(String tags)</code> <code>String getSetName()</code> <code>Integer getSetId()</code> <code>IResource[] getResources()</code> <code>void addResource(Integer resId)</code> <code>void removeResource(Integer resId)</code> <code>void refresh()</code> <code>void setSetName(String newName)</code> |

Table C.1: Methods associated with RDT entities (IResource, ISet)

| |
|--|
| IPProject |
| String getTags() void setTags(String tags) String getProjectName() Integer getProjectId() void setProjectName(String newName) |
| ISession |
| String getTags() void setTags(String tags) String getSessionName() Integer getSessionId() IPProject getProject() ISet getSet() String getConfig() void setSessionName(String newName) void setProject(Integer newProject) void setSet(Integer newSet) void setConfig(String newConfig) |
| ITask |
| String getTaskName() Integer getTaskId() String getTaskType() String getTaskStatus() String getTaskOutput() void refresh() ISession getSession() IResource getResource() |

Table C.2: Methods associated with RDT entities (IPProject, ISession, ITask)

| |
|---|
| ResourceManager |
| void refresh() IResource newResource(String resName, int res_id) void destroyResource(Integer id) IResource[] getResources() IResource getResource(Integer id) IResource getResource(String resName) |
| SetManager |
| void refresh() ISet newSet(String setName) void destroySet(Integer id) ISet[] getSets() ISet getSet(Integer id) ISet getSet(String setName) |
| ProjectManager |
| void refresh() IProject newProject (String projectName) void destroyProject(Integer id) IProject[] getProjects() IProject getProject(Integer id) IProject getProject(String projName) |
| SessionManager |
| void refresh() ISession newSession (String sessionName, int proj_id, int set_id) void destroySession(Integer id) ISession[] getSessions() ISession getSession(Integer id) ISession getSession(String sessName) |
| TaskManager |
| void refresh() void destroyTask(Integer id) void destroyAllTasks() ITask[] getTasks() ITask getTask(Integer id) ITask getTask(String taskName) |

Table C.3: Methods associated with RDT entity managers

Appendix D

GridDebug Debug Clients

```
1 package org.globus.monash.clients;
2
3 // Various Java import statements ...
4
5 public class CLIDebugClient implements DebugQNames, ↵
6     NotifyCallback {
7     static { Util.registerTransport(); }
8     static DebugPortType debug;
9
10    public static void main(String[] args) {
11        CLIDebugClient client = new CLIDebugClient();
12        NotificationConsumerManager consumer = null;
13
14        DebugFactoryServiceAddressingLocator factoryLocator = ↵
15            new DebugFactoryServiceAddressingLocator();
16        DebugServiceAddressingLocator instanceLocator = new ↵
17            DebugServiceAddressingLocator();
18
19        try {
20            String factoryURI = "https://127.0.0.1:8443/wsrf/↵
21                services/monash/core/DebugFactoryService";
22            EndpointReferenceType factoryEPR, instanceEPR;
23            DebugFactoryPortType debugFactory;
24
25            // Get factory PortType
26            factoryEPR = new EndpointReferenceType();
```

```

23     factoryEPR.setAddress(new Address(factoryURI));
24     debugFactory = factoryLocator.←
        getDebugFactoryPortTypePort(factoryEPR);
25
26     // Create resource and get the end-point reference
27     CreateResourceResponse createResponse = debugFactory
28         .createResource(new CreateResource());
29     instanceEPR = createResponse.getEndpointReference();
30
31     // Get instance PortType
32     debug = instanceLocator.getDebugPortTypePort(←
        instanceEPR);
33
34     consumer = NotificationConsumerManager.getInstance();
35     consumer.startListening();
36
37     EndpointReferenceType consumerEPR = consumer.←
        createNotificationConsumer(client);
38
39     // Subscribe for notifications (program output)
40     Subscribe prgOutRequest = new Subscribe();
41     prgOutRequest.setUseNotify(Boolean.TRUE);
42     prgOutRequest.setConsumerReference(consumerEPR);
43     TopicExpressionType prgOutTopExp = new ←
        TopicExpressionType();
44     prgOutTopExp.setDialect(WSNConstants.←
        SIMPLE_TOPIC_DIALECT);
45     prgOutTopExp.setValue(DebugQNames.RP_PROGRAMOUTPUT);
46     prgOutRequest.setTopicExpression(prgOutTopExp);
47     SubscribeResponse prgOutResponse = debug.subscribe(←
        prgOutRequest);
48
49     // Subscribe for notifications (debugger output)
50     Subscribe dbgOutRequest = new Subscribe();
51     dbgOutRequest.setUseNotify(Boolean.TRUE);
52     dbgOutRequest.setConsumerReference(consumerEPR);
53     TopicExpressionType dbgOutTopExp = new ←
        TopicExpressionType();

```

```

54     dbgOutTopExp.setDialect(WSNConstants.↵
        SIMPLE_TOPIC_DIALECT);
55     dbgOutTopExp.setValue(DebugQNames.RP_DEBUGGEROUTPUT);
56     dbgOutRequest.setTopicExpression(dbgOutTopExp);
57     SubscribeResponse dbgOutResponse = debug.subscribe(↵
        dbgOutRequest);
58
59     // Initialize debugging
60     debug.debugInit(new DebugInit("NONE", "NONE"));
61
62     String line;
63     BufferedReader in = new BufferedReader (new ↵
        InputStreamReader(System.in));
64     System.out.println("\nCLI GridDebug Debugger:");
65
66     // Process user input
67     System.out.print("::> ");
68     while((line = in.readLine()) != null) {
69         if (line.equals("exit")) break;
70         processInput(line);
71         System.out.print("::> ");
72     }
73     in.close();
74
75     System.out.println("Exiting...");
76
77     debug.debugExit(new DebugExit());
78
79     SubscriptionManagerServiceAddressingLocator sLocator =↵
        new SubscriptionManagerServiceAddressingLocator()↵
        ;
80     sLocator.getSubscriptionManagerPort(prgOutResponse.↵
        getSubscriptionReference()).destroy(new Destroy())↵
        ;
81     sLocator.getSubscriptionManagerPort(dbgOutResponse.↵
        getSubscriptionReference()).destroy(new Destroy())↵
        ;
82
83     if (consumer != null) consumer.stopListening();

```

```
84
85     // Destroy resource
86     debug.destroy(new Destroy());
87 } catch (Exception e) {
88     e.printStackTrace();
89 }
90 }
91
92 // Input processing method
93 public static void processInput(String input) throws ↵
94     RemoteException {
95     String[] result = input.split("\\s");
96
97     if (result == null || result.length == 0)
98         return;
99
100     if (result[0].equals("load")) {
101         if (result.length != 2) {
102             System.out.println("Wrong command");
103             return;
104         }
105         debug.debugLoad(new DebugLoad(1, result[1]));
106
107     } else if (result[0].equals("break")) {
108         if (result.length != 2) {
109             System.out.println("Wrong command");
110             return;
111         }
112         debug.debugBreakpoint(new DebugBreakpoint(null, 0, ↵
113             result[1]));
114
115     } else if (result[0].equals("run")) {
116         debug.debugRun(new DebugRun());
117
118     } else if (result[0].equals("cont")) {
119         debug.debugCont(new DebugCont());
120
121     } else if (result[0].equals("stepFinish")) {
122         debug.debugStepFinish(new DebugStepFinish());
```

```

121
122     } else if (result[0].equals("step")) {
123         debug.debugStep(new DebugStep());
124
125     } else if (result[0].equals("stepOver")) {
126         debug.debugStepOver(new DebugStepOver());
127
128     } else {
129         System.out.println("Wrong command");
130     }
131 }
132
133 // Notification callback method
134 public void deliver(List topicPath, EndpointReferenceType ↵
135     producer, Object message) {
136     ResourcePropertyValueChangeNotificationType ↵
137         changeMessage = ((↵
138             ResourcePropertyValueChangeNotificationElementType) ↵
139             message).getResourcePropertyValueChangeNotification↵
140             ();
141
142     if(changeMessage != null) {
143         if (topicPath.get(0).equals(DebugQNames.↵
144             RP_DEBUGGEROUTPUT))
145             System.out.print("\nDebugger Out: " + changeMessage.↵
146                 getNewValue().get_any()[0].getValue());
147         else if (topicPath.get(0).equals(DebugQNames.↵
148             RP_PROGRAMOUTPUT))
149             System.out.print("\nProgram Out: " + changeMessage.↵
150                 getNewValue().get_any()[0].getValue());
151     }
152
153     synchronized (this) {
154         notify();
155     }
156 }
157 }
158 }

```

Listing D.1: A command-line interface (CLI) GridDebug debugger

```
1 # Various Jython import statements ...
2
3 # Set debug factories
4 factoryURI1 = "https://east-globus.enterprisegrid.edu.au↵
      :8443/wsrf/services/monash/core/DebugFactoryService"
5 factoryURI2 = "https://west-globus.enterprisegrid.edu.au↵
      :8443/wsrf/services/monash/core/DebugFactoryService"
6
7 Util.registerTransport()
8 factoryLocator = DebugFactoryServiceAddressingLocator()
9 instanceLocator = DebugServiceAddressingLocator()
10
11 # Get factory PortType
12 factEPR1 = EndpointReferenceType()
13 factEPR2 = EndpointReferenceType()
14 factEPR1.setAddress(Address(factoryURI1))
15 factEPR2.setAddress(Address(factoryURI2))
16 dFact1 = factoryLocator.getDebugFactoryPortTypePort(factEPR1↵
      )
17 dFact2 = factoryLocator.getDebugFactoryPortTypePort(factEPR2↵
      )
18
19 # Create resource and get end-point reference
20 createResponse1 = dFact1.createResource(CreateResource())
21 createResponse2 = dFact2.createResource(CreateResource())
22 instanceEPR1 = createResponse1.getEndpointReference()
23 instanceEPR2 = createResponse2.getEndpointReference()
24
25 # Get instance PortType
26 debug1 = instanceLocator.getDebugPortTypePort(instanceEPR1)
27 debug2 = instanceLocator.getDebugPortTypePort(instanceEPR2)
28
29 program_path = "/home/donny/tmp/test/Jacobi"
30 bp_loc = "387"
31 variable_name = "stat"
32
33 # Initialize the debugger
34 debug1.debugInit(DebugInit("NONE", "NONE"))
35 debug2.debugInit(DebugInit("NONE", "NONE"))
```



```

36
37 # Load the binary
38 debug1.debugLoad(DebugLoad(1, program_path))
39 debug2.debugLoad(DebugLoad(1, program_path))
40
41 # Set the breakpoint
42 debug1.debugBreakpoint(DebugBreakpoint(None, 0, bp_loc))
43 debug2.debugBreakpoint(DebugBreakpoint(None, 0, bp_loc))
44
45 # Execute the program
46 debug1.debugRun(DebugRun())
47 debug2.debugRun(DebugRun())
48
49 # We are stopping at line "bp_loc" now
50
51 # Retrieve the variable
52 var1 = debug1.debugPrint(variable_name)
53 var2 = debug2.debugPrint(variable_name)
54
55 # Continue the program execution
56 debug1.debugCont(DebugCont())
57 debug2.debugCont(DebugCont())
58
59 # Quit the debugger
60 debug1.debugExit(DebugExit())
61 debug2.debugExit(DebugExit())
62
63 # Destroy the debugger
64 debug1.destroy(Destroy())
65 debug2.destroy(Destroy())
66
67 if var1 == var2:
68     print "They terminate at the same iteration: " + var1
69 else:
70     print "They do not terminate at the same iteration: " + ↵
        var1 + " and " + var2

```

Listing D.2: A Jython script that performs a comparison between variables

Bibliography

David Abramson. Applications Development for the Computational Grid.
In *APWEB 2006: Proceedings of the Eighth Asia Pacific Web Conference*,
January 2006.

David Abramson and Rok Sasic. A Debugging Tool for Software Evolution.
In *CASE 1995: Proceedings of the Seventh International Workshop on
Computer-Aided Software Engineering*, July 1995.

David Abramson, Jon Giddy, and Lew Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?
In *IPDPS 2000: Proceedings of the 14th International Parallel and Distributed Processing Symposium*, May 2000a.

David Abramson, Andrew Lewis, and Tom Peachey. Nimrod/O: A Tool for Automatic Design Optimization using Parallel and Distributed Systems.
In *ICA3PP 2000: Proceedings of the 4th International Conference on Algorithms and Architecture for Parallel Processing*, December 2000b.

David Abramson, Celine Amoreira, Kim Baldridge, Laura Berstis, Chris Kondrick, and Tom Peachey. A Flexible Grid Framework for Automatic Protein-Ligand Docking. In *e-Science 2006: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, December 2006a.

David Abramson, Amanda Lynch, Hiroshi Takemiya, Yusuke Tanimura, Susumu Date, Haruki Nakamura, Karpjoo Jeong, Suntae Hwang, Ji Zhu,

- Zhong hua Lu, Celine Amoreira, Kim Baldridge, Hurng-Chun Lee, Chi-Wei Wang, Horng-Liang Shih, Tomas Molina, Wilfred W. Li, and Peter A. Arzberger. Deploying Scientific Applications to the PRAGMA Grid Testbed: Strategies and Lessons. In *CCGrid 2006: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, May 2006b.
- David Abramson, Tom Peachey, and Andrew Lewis. Model Optimization and Parameter Estimation with Nimrod/O. In *ICCS 2006: Proceedings of the Sixth International Conference on Computational Science*, May 2006c.
- Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language*. Addison-Wesley, 1988.
- Jay Alameda, Marcus Christie, Geoffrey Fox, Joe Futrelle, Dennis Gannon, Mihael Hategan, Gopi Kandaswamy, Gregor von Laszewski, Mehmet A. Nacar, Marlon Pierce, Eric Roberts, Charles Severance, and Mary Thomas. The Open Grid Computing Environments Collaboration: Portlets and Services for Science Gateways. *Concurrency and Computation: Practice and Experience*, 19(6):921–942, April 2007.
- Marco Aldinucci, Massimo Coppola, Sonia Campa, Marco Danelutto, Marco Vanneschi, and Corrado Zoccolo. Structured Implementation of Component-Based Grid Programming Environments. In Vladimir Getov, Domenico Laforenza, and Alexander Reinefeld, editors, *Future Generation Grids*, pages 217–239. Springer, 2005.
- Marco Aldinucci, Massimo Coppola, Marco Vanneschi, Corrado Zoccolo, and Marco Danelutto. ASSIST as a Research Framework for High-Performance Grid Programming Environments. In Jose Cunha and Omer Rana, editors, *Grid Computing: Software Environments and Tools*, chapter 10, pages 230–256. Springer, 2006.
- Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke. Secure, Efficient Data Transport and Replica Management for

- High-Performance Data-Intensive Computing. In *MSS 2001: Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems*, April 2001.
- Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke. Data Management and Transfer in High-Performance Computational Grid Environments. *Parallel Computing*, 28(5), May 2002.
- Gabrielle Allen, Kelly Davis, Tom Goodale, Andrei Hutanu, Hartmut Kaiser, Thilo Kielmann, Andre Merzky, Rob van Nieuwpoort, Alexander Reinefeld, Florian Schintke, Thorsten Schutt, Ed Seidel, and Brygg Ullmer. The Grid Application Toolkit: Toward Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3), March 2005.
- Jim Amsden. Levels Of Integration: Five ways you can integrate with the Eclipse Platform, March 2001. <http://www.eclipse.org/articles/Article-Levels-Of-Integration/levels-of-integration.html> [3 September 2007].
- Anjuta. Anjuta Integrated Development Environment, 2008. <http://anjuta.sourceforge.net/> [22 January 2008].
- Ant. Apache Ant, 2007. <http://ant.apache.org/> [28 August 2007].
- ANTLR. ANTLR Parser Generator, 2008. <http://www.antlr.org/> [30 June 2008].
- Apache HTTPD. The Apache HTTP Server Project, 2008. <http://httpd.apache.org/> [3 June 2008].
- AppleScript. Apple Developer Connection - AppleScript, 2008. <http://developer.apple.com/applescript/> [22 January 2008].
- APR. Apache Portable Runtime, 2008. <http://apr.apache.org/> [11 March 2008].
- Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar, and Srikumar Venugopal. Global Grids and Software Toolkits: A Study of Four

- Grid Middleware Technologies. In Laurence T. Yang and Minyi Guo, editors, *High-Performance Computing: Paradigm and Infrastructure*, chapter 22. John Wiley and Sons, 2005.
- Malcolm Atkinson. Rationale for Choosing the Open Grid Services Architecture. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 7. John Wiley and Sons, 2003.
- Automator. Introduction to Automator Programming Guide, 2008. <http://developer.apple.com/documentation/AppleApplications/Conceptual/AutomatorConcepts/> [22 January 2008].
- Lerina Aversano, Massimiliano Di Penta, and Ira Baxter. Handling Preprocessor-Conditioned Declarations. In *SCAM 2002: Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation*, October 2002.
- AWS. Amazon Web Services, 2009. <http://aws.amazon.com/> [11 March 2009].
- Henri Bal, Henri Casanova, Jack Dongarra, and Satoshi Matsuoka. Application-Level Tools. In Ian Foster and Carl Kesselman, editors, *The Grid 2: Blueprint for a New Computing Infrastructure*, chapter 24. Morgan Kaufmann, 2003.
- Kim Baldridge and Philip E. Bourne. The New Biology and the Grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 40. John Wiley and Sons, 2003.
- Susanne M. Balle and Robert T. Hood. GGF UPDT User Development Tools Survey. *OGF Document Series*, October 2004. <http://www.ogf.org/documents/GFD.33.pdf> [11 August 2007].
- Richard Barker. *CASE Method: Entity Relationship Modelling*. Addison-Wesley, 1990.

- Bazaar. Bazaar Version Control, 2008. <http://bazaar-vcs.org/> [30 June 2008].
- Rudiger Berlich, Marcel Kunze, and Kilian Schwarz. Grid Computing in Europe: From Research to Deployment. In *AusGrid 2005: Proceedings of the 3rd Australasian Workshop on Grid Computing and e-Research*, January 2005.
- Fran Berman and Tony Hey. The Scientific Imperative. In Ian Foster and Carl Kesselman, editors, *The Grid 2: Blueprint for a New Computing Infrastructure*, chapter 2. Morgan Kaufmann, 2003.
- Binutils. GNU Binutils, 2008. <http://www.gnu.org/software/binutils/> [30 June 2008].
- Bison. Bison - GNU Parser Generator, 2008. <http://www.gnu.org/software/bison/> [30 June 2008].
- Caspar Boekhoudt. The Big Bang Theory of IDEs. *ACM Queue*, 1(7), 2003.
- Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, and Eduardo Gomez-Sanchez. Grid Characteristics and Uses: A Grid Definition. In *AxGrids 2003: Proceedings of the First European Across Grids Conference*, February 2003.
- Michael Brady, David Gavaghan, Andrew Simpson, Miguel M. Parada, and Ralph Highnam. eDiamond: a Grid-enabled Federated Database of Annotated Mammograms. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 41. John Wiley and Sons, 2003.
- Ilja N. Bronshtein, Konstantin A. Semendyayev, Gerhard Musiol, and Heiner Muehlig. *Handbook of Mathematics*. Springer, fifth edition, 2007.
- Alan Brown. An introduction to Model Driven Architecture. Part I: MDA and Today's Systems, 2004. <http://www.ibm.com/developerworks/rational/library/3100.html> [10 January 2008].

- Alan Brown and John McDermid. Learning from IPSE's Mistakes. *IEEE Software*, 9(2), 1992.
- Julian J. Bunn and Harvey B. Newman. Data-intensive Grids for High-energy Physics. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 39. John Wiley and Sons, 2003.
- Randy Butler, Douglas Engert, Ian Foster, Steven Tuecke, John Volmer, and Carl Kesselman. A National-Scale Authentication Infrastructure. *IEEE Computer*, 33(12), December 2000.
- Carbon. Apple Developer Connection - Carbon, 2008. <http://developer.apple.com/carbon/> [22 January 2008].
- Denis Caromel, Christian Delbe, Alexandre di Costanzo, and Mario Leyton. ProActive: An Integrated Platform for Programming and Running Applications on Grids and P2P Systems. *Computational Methods in Science and Technology*, 12(1), 2006.
- Henri Casanova and Fran Berman. Parameter Sweeps on the Grid with APST. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 33. John Wiley and Sons, 2003.
- Albert F. Case. Computer-aided software engineering (CASE): technology for improving software development productivity. *ACM SIGMIS Database*, 17(1), September 1985.
- Philip Chan and David Abramson. Netfiles: An Enhanced Stream-based Communication Mechanism. In *ISHPC 2005: Proceedings of the Sixth International Symposium on High Performance Computing*, September 2005.
- David Churches, Gabor Gombas, Andrew Harrison, Jason Maassen, Craig Robinson, Matthew Shields, Ian Taylor, and Ian Wang. Programming Scientific and Distributed Workflow with Triana Services. *Concurrency and Computation: Practice and Experience*, 18(10):1021–1037, August 2006.

- Eric Clayberg and Dan Rubel. *Eclipse: Building Commercial-Quality Plugins*. Addison-Wesley, second edition, 2006.
- Cocoa. Apple Developer Connection - Cocoa, 2008. <http://developer.apple.com/cocoa/> [22 January 2008].
- Condor. Condor High Throughput Computing, 2007. <http://www.cs.wisc.edu/condor/> [14 September 2007].
- CVS. CVS - Concurrent Versions System, 2006. <http://www.nongnu.org/cvs/> [12 June 2008].
- Karl Czajkowski, Ian Foster, Nicholas Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A Resource Management Architecture for Metacomputing Systems. In *JSSPP 1998: Proceedings of the Fourth Workshop on Job Scheduling Strategies for Parallel Processing in conjunction with IPPS/SPDP 1998*, March 1998.
- Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselman. Grid Information Services for Distributed Resource Sharing. In *HPDC 2001: Proceedings of the 10th International Symposium on High Performance Distributed Computing*, August 2001.
- Jim des Rivieres and Wayne Beaton. Eclipse Platform Technical Overview, 2006. <http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html> [29 August 2007].
- Jim des Rivieres and John Wiegand. Eclipse: A platform for integrating development tools. *IBM Systems Journal*, 43(2), 2004.
- Prashant Deva. Folding in Eclipse Text Editors, March 2005. <http://www.eclipse.org/articles/Article-Folding-in-Eclipse-Text-Editors/folding.html> [15 October 2008].
- Tim Dierks and Eric Rescorla. RFC 4346 - The Transport Layer Security (TLS) Protocol Version 1.1, 2006. <http://tools.ietf.org/html/rfc4346> [3 June 2008].

- distcc. distcc: a fast, free distributed C/C++ compiler, 2007. <http://distcc.samba.org/> [27 August 2007].
- DTrace. DTrace at OpenSolaris.org, 2008. <http://opensolaris.org/os/community/dtrace/> [30 June 2008].
- Eclipse. Eclipse.org Home, 2007. <http://www.eclipse.org/> [29 August 2007].
- Emacs. GNU Emacs - GNU Project - Free Software Foundation (FSF), 2008. <http://www.gnu.org/software/emacs/> [30 June 2008].
- Enterprise Grid. Message Lab - Enterprise Grid, 2008. <http://www.enterprisegrid.edu.au/> [17 November 2008].
- Michael Ernst, Greg Badros, and David Notkin. An Empirical Analysis of C Preprocessor Use. *IEEE Transactions on Software Engineering*, 28(12), 2002.
- Dietmar W. Erwin and David F. Snelling. UNICORE: A Grid Computing Environment. In *Euro-Par 2001: Proceedings of the 7th International Euro-Par Conference*, August 2001.
- Facebook. Facebook Home, 2009. <http://www.facebook.com/> [4 February 2009].
- Noel Faux, Anthony Beitz, Mark Bate, Abdullah A. Amin, Ian Atkinson, Colin Enticott, Khalid Mahmood, Matthew Swift, Andrew Treloar, David Abramson, James C. Whisstock, and Ashley M. Buckle. eResearch Solutions for High Throughput Structural Biology. In *e-Science 2007: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, December 2007.
- Obie Fernandez. *The Rails Way*. Addison-Wesley, 2007.
- Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

Steven Fitzgerald, Ian Foster, Carl Kesselman, Gregor von Laszewski, Warren Smith, and Steven Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *HPDC 1997: Proceedings of the 6th International Symposium on High Performance Distributed Computing*, August 1997.

Folding@Home. Folding@Home Site, 2009. <http://folding.stanford.edu/> [12 February 2009].

Ian Foster. What is the Grid? A Three Point Checklist, 2002. <http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf> [4 September 2007].

Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *NPC 2005: Proceedings of the IFIP International Conference on Network and Parallel Computing*, December 2005a.

Ian Foster. A Globus Primer, August 2005b. http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf [2 February 2008].

Ian Foster and Adriana Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *IPTPS 2003: Proceedings of the Second International Workshop on Peer-to-Peer Systems*, February 2003.

Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications*, 11(2), 1997.

Ian Foster and Carl Kesselman. Computational Grids. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 2. Morgan Kaufmann, 1998a.

Ian Foster and Carl Kesselman. The Globus Project: A Status Report. In *HCW 1998: Proceedings of the Seventh Heterogeneous Computing Workshop*, March 1998b.

- Ian Foster and Carl Kesselman. Concepts and Architecture. In Ian Foster and Carl Kesselman, editors, *The Grid 2: Blueprint for a New Computing Infrastructure*, chapter 4. Morgan Kaufmann, 2003.
- Ian Foster and Steve Tuecke. Describing the Elephant: the Different Faces of IT as Service. *ACM Queue*, 3(6), 2005.
- Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A Security Architecture for Computational Grids. In *CCS 1998: Proceedings of the 5th ACM Conference on Computer and Communications Security*, November 1998.
- Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The International Journal of Supercomputer Applications*, 15(3), August 2001.
- Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The Physiology of the Grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 8. John Wiley and Sons, 2003a.
- Ian Foster, Carl Kesselman, and Steven Tuecke. The Open Grid Services Architecture. In Ian Foster and Carl Kesselman, editors, *The Grid 2: Blueprint for a New Computing Infrastructure*, chapter 17. Morgan Kaufmann, 2003b.
- Ian Foster, Hiro Kishimoto, Andreas Savva, Dave Berry, Abdeslem Djaoui, Andrew Grimshaw, Bill Horn, Fred Maciel, Frank Siebenlist, Ravi Subramaniam, Jem Treadwell, and Jeffrin Von Reich. The Open Grid Services Architecture, Version 1.5. *OGF Document Series*, July 2006. <http://www.ogf.org/documents/GFD.80.pdf> [30 January 2008].
- Amy Fowler. A Swing Architecture Overview, 2007. <http://java.sun.com/products/jfc/tsc/articles/architecture/> [28 August 2007].
- Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.

- Geoffrey Fox, Dennis Gannon, and Mary Thomas. Overview of Grid Computing Environments. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 20. John Wiley and Sons, 2003.
- Joan M. Francioni and Cherri M. Pancake. A Debugging Standard for High-Performance Computing. *Scientific Programming*, 8(2), 2000.
- David Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley and Sons, 2003.
- Jeremy G. Frey, Mark Bradley, Jonathan W. Essex, Michael B. Hursthouse, Susan M. Lewis, Michael M. Luck, Luc Moreau, Dave C. De Roure, Mike Surridge, and Alan H. Welsh. Combinatorial Chemistry and the Grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 42. John Wiley and Sons, 2003.
- Thomas Frieze, Matthew Smith, and Bernd Freisleben. Grid Development Tools for Eclipse. In *eTX 2006: Proceedings of the Eclipse Technology Exchange Workshop in conjunction with ECOOP 2006*, July 2006a.
- Thomas Frieze, Matthew Smith, and Bernd Freisleben. GDT: A Toolkit for Grid Service Development. In *GSEM 2006: Proceedings of the Third International Conference on Grid Service Engineering and Management*, September 2006b.
- Alfonso Fuggetta. A Classification of CASE Technology. *IEEE Computer*, 26(12), December 1993.
- g-Eclipse. g-Eclipse - Access the Grid, 2009. <http://www.geclipse.org/> [11 March 2009].
- Fabrizio Gagliardi, Bob Jones, Francois Grey, Marc-Eliaen Begin, and Matti Heikkurinen. Building an infrastructure for scientific Grid computing: status and goals of the EGEE project. *Philosophical Transactions of the Royal*

Society A: Mathematical, Physical and Engineering Sciences, 363(1833), August 2005.

Erich Gamma and Kent Beck. *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*. Addison-Wesley, 2003.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

Dennis Gannon, Kenneth Chiu, Madhusudhan Govindaraju, and Aleksander Slominski. An Analysis of The Open Grid Services Architecture, 2002. <http://www.extreme.indiana.edu/~gannon/OGSAanalysis3.pdf> [30 January 2008].

Dennis Gannon, Sriram Krishnan, Liang Fang, Gopi Kandaswamy, Yogesh Simmhan, and Aleksander Slominski. On Building Parallel and Grid Applications: Component Technology and Distributed Services. In *CLADE 2004: Proceedings of the Workshop on Challenges of Large Applications in Distributed Environments in conjunction with HPDC 2004*, June 2004.

Jesse Garrett. Ajax: A New Approach to Web Applications, 2005. <http://www.adaptivepath.com/ideas/essays/archives/000385.php> [26 May 2008].

GCC. GCC, the GNU Compiler Collection, 2007. <http://gcc.gnu.org/> [27 August 2007].

GDB. GDB: The GNU Project Debugger, 2007. <http://sourceware.org/gdb/> [27 August 2007].

Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidyalingam S. Sunderam. *PVM: Parallel Virtual Machine*. MIT Press, November 1994.

Glade. Glade User Interface Builder, 2008. <http://glade.gnome.org/> [22 January 2008].

Robert L. Glass. The realities of software technology payoffs. *Communications of the ACM*, 42(2), February 1999.

GLib. GLib Reference Manual, 2008. <http://library.gnome.org/devel/glib/stable/> [11 March 2008].

gLite. gLite - Lightweight Middleware for Grid Computing, 2008. <http://www.glite.org/> [2 February 2008].

GNOME. GNOME: The Free Software Desktop Project, 2008. <http://www.gnome.org/> [22 January 2008].

Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Masso, Thomas Radke, Edward Seidel, and John Shalf. The Cactus Framework and Toolkit: Design and Applications. In *VecPar 2002: Proceedings of the 5th International Meeting on Vector and Parallel Processing*, June 2002.

Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Gregor von Laszewski, Craig Lee, Andre Merzky, Hrabri Rajic, and John Shalf. SAGA: A Simple API for Grid Applications, High-Level Application Programming on the Grid. *Computational Methods in Science and Technology*, 12(1), 2006.

GRIA. Service Oriented Collaborations for Industry and Commerce - GRIA, 2009. <http://www.gria.org/> [11 March 2009].

Grid Workflow Forum. Grid Workflow Projects, 2008. <http://www.gridworkflow.org/snips/gridworkflow/space/Projects> [9 February 2008].

GridLab. GridLab: A Grid Application Toolkit and Testbed, 2008. <http://www.gridlab.org/> [18 February 2008].

GridPort. The GridPort Toolkit, 2008. <http://gridport.net/main/> [18 February 2008].

- GridRPC-WG. The Grid Remote Procedure Call Working Group (GridRPC-WG), 2008. <https://forge.gridforum.org/projects/gridrpc-wg/> [20 February 2008].
- GridSphere. The GridSphere Portal Framework Project, 2008. <http://www.gridsphere.org/> [18 February 2008].
- GROMACS. GROMACS: Fast, Free, and Flexible MD, 2009. <http://www.gromacs.org/> [12 February 2009].
- Groovy. Groovy Home, 2008. <http://groovy.codehaus.org/> [13 June 2008].
- William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir. *MPI: The Complete Reference, Volume 2: The MPI-2 Extensions*. MIT Press, September 1998.
- Olivier Gruber, B. J. Hargrave, Jeff McAffer, Pascal Rapicault, and Thomas Watson. The Eclipse 3.0 platform: Adopting OSGi technology. *IBM Systems Journal*, 44(2), 2005.
- GT2. Globus 2.4 Overview, 2007. <http://globus.org/toolkit/docs/2.4/admin/guide-overview.html> [10 September 2007].
- GT4IDE. Globus Service Build Tools - GT4IDE, 2008. <http://gsbt.sourceforge.net/content/view/12/29/> [9 January 2008].
- GTK+. GTK+: The GIMP Toolkit, 2008. <http://www.gtk.org/> [22 January 2008].
- Shannon Hastings, Scott Oster, Stephen Langella, David Ervin, Tahsin Kurc, and Joel Saltz. Introduce: An Open Source Toolkit for Rapid Development of Strongly Typed Grid Services. *Journal of Grid Computing*, 5(4), December 2007.
- Tony Hey and Anne Trefethen. e-Science and its implications. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 361(1809), August 2003.

- David Hollingsworth. Workflow Management Coalition: The Workflow Reference Model, 1995. <http://www.wfmc.org/standards/docs/tc003v11.pdf> [12 February 2008].
- HPDF. HPD Version 1 Standard: Command Interface for Parallel Debuggers, 1998. http://web.archive.org/web/20010702151435rn_1/www.ptools.org/hpdf/draft/ [19 March 2008].
- IBM. IBM developerWorks : Standards and Web services, 2008. <http://www.ibm.com/developerworks/webservices/standards/> [25 January 2008].
- ICC. Intel Compilers, 2008. <http://www.intel.com/cd/software/products/asm-na/eng/compilers/284132.htm> [30 June 2008].
- IDB. Intel Debugger (IDB) Manual, 2008. http://www.intel.com/software/products/compilers/docs/linux/idb_manual_1.html [22 July 2008].
- IEEE. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990, 1990. http://standards.ieee.org/reading/ieee/std_public/description/se/610.12-1990_desc.html [16 January 2008].
- IronRuby. Microsoft IronRuby, 2008. <http://www.ironruby.net/> [30 May 2008].
- Java Community Process. JSR (Java Specification Request) 168: Portlet Specification, 2008. <http://jcp.org/en/jsr/detail?id=168> [18 February 2008].
- Javadoc. Javadoc Tool Home Page, 2008. <http://java.sun.com/j2se/javadoc/> [30 June 2008].
- Shantenu Jha, Hartmut Kaiser, Andre Merzky, and Ole Weidner. Grid Interoperability at the Application Level Using SAGA. In *IGIIW 2007: Proceedings of the International Grid Interoperability and Interoperation Workshop in conjunction with e-Science 2007*, December 2007.

- Stephen C. Johnson. YACC: Yet Another Compiler-Compiler, 1979. <http://www2.informatik.uni-erlangen.de/Lehre/WS200304/Compilerbau/Uebungen/yacc.pdf> [30 June 2008].
- Bob Jones. An Overview of the EGEE Project. In *Proceedings of the Sixth Thematic Workshop of the EU Network of Excellence DELOS*, June 2004.
- Simon Josefsson. RFC 4648 - The Base16, Base32, and Base64 Data Encodings, 2006. <http://tools.ietf.org/html/rfc4648> [3 June 2008].
- JRuby. JRuby Home, 2008. <http://jruby.codehaus.org/> [30 May 2008].
- Jython. The Jython Project, 2008. <http://www.jython.org/Project/> [30 December 2008].
- Peter Kacsuk. Systematic Macrostep Debugging of Message Passing Parallel Programs. *Journal of Future Generation Computer Systems*, 16(6), 2000.
- Peter Kacsuk, Gabor Dozsa, Jozsef Kovacs, Robert Lovas, Norbert Podhorszki, Zoltan Balaton, and Gabor Gombas. P-GRADE: A Grid Programming Environment. *Journal of Grid Computing*, 1(2), June 2003.
- Nicholas Karonis, Brian Toonen, and Ian Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5), May 2003.
- Kate. Kate - KDE Advanced Text Editor, 2008. <http://www.kate-editor.org/> [22 January 2008].
- KDevelop. KDevelop - an Integrated Development Environment, 2008. <http://kdevelop.org/> [22 January 2008].
- Rainer Keller, Bettina Krammer, Matthias Mueller, Michael Resch, and Edgar Gabriel. MPI Development Tools and Applications for the Grid. In *Proceedings of the Workshop on Grid Applications and Programming Tools in conjunction with GGF8*, June 2003.

- John Kemeny and Thomas Kurtz. BASIC, 1964. http://www.bitsavers.org/pdf/dartmouth/BASIC_Oct64.pdf [22 January 2008].
- Thilo Kielmann, Rutger Hofman, Henri Bal, Aske Plaat, and Raoul Bhoedjang. MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems. In *PPoPP 1999: Proceedings of the Seventh ACM Symposium on Principles and Practice of Parallel Programming*, May 1999.
- Thilo Kielmann, Andre Merzky, Henri Bal, Françoise Baude, Denis Carmel, and Fabrice Huet. Grid Application Programming Environments. In Vladimir Getov, Domenico Laforenza, and Alexander Reinefeld, editors, *Future Generation Grids*, pages 283–306. Springer, 2005.
- Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.
- Rex Bryan Kline and Ahmed Seffah. Evaluation of integrated software development environments: challenges and results from three empirical studies. *International Journal of Human-Computer Studies*, 63(6), December 2005.
- Sriram Krishnan and Dennis Gannon. XCAT3: A Framework for CCA Components as OGSA Services. In *HIPS 2004: Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments in conjunction with IPDPS 2004*, April 2004.
- Vincent Kruskal. A Blast from the Past: Using P-EDIT for Multidimensional Editing. In *Proceedings of the Workshop on Multi-Dimensional Separation of Concerns in Software Engineering in conjunction with ICSE 2000*, June 2000.
- Bernt Kullbach and Volker Riedinger. Folding: An Approach to Enable Program Understanding of Preprocessed Languages. In *WCRE 2001: Proceedings of the Eighth Working Conference on Reverse Engineering*, 2001.
- Donny Kurniawan and David Abramson. Worqbench: An Integrated Framework for e-Science Application Development. In *e-Science 2006: Proceed-*

ings of the Second IEEE International Conference on e-Science and Grid Computing, December 2006.

Donny Kurniawan and David Abramson. A WSRF-Compliant Debugger for Grid Applications. In *IPDPS 2007: Proceedings of the 21th International Parallel and Distributed Processing Symposium*, March 2007a.

Donny Kurniawan and David Abramson. An Integrated Grid Development Environment in Eclipse. In *e-Science 2007: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, December 2007b.

Donny Kurniawan and David Abramson. An IDE Framework for Grid Application Development. In *Grid 2008: Proceedings of the Ninth International Workshop on Grid Computing*, September 2008.

Thomas Kurtz. BASIC. In Richard Wexelblat, editor, *History of Programming Languages*, pages 515–537. Academic Press, 1981.

Domenico Laforenza. Grid Programming: Some Indications Where We are Headed. *Parallel Computing*, 28(12), December 2002.

Erwin Laure, Frederic Hemmer, Francesco Prelz, Stefano Beco, Steve Fisher, Miron Livny, Leanne Guy, Maite Barroso, Predrag Buncic, Peter Kunszt, Alberto Di Meglio, Alberto Aimar, Ake Edlund, David Groep, Fabrizio Pacini, Massimo Sgaravatto, and Olle Mulmo. Middleware for the Next Generation Grid Infrastructure. In *CHEP 2004: Proceedings of the International Conference on Computing in High Energy Physics*, 2004.

Erwin Laure, Steve Fisher, Akos Frohner, Claudio Grandi, Peter Kunszt, Ales Krenek, Olle Mulmo, Fabrizio Pacini, Francesco Prelz, John White, Maite Barroso, Predrag Buncic, Frederic Hemmer, Alberto Di Meglio, and Ake Edlund. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1), 2006.

LCG. The LHC Computing Grid Project, 2007. <http://lcg.web.cern.ch/LCG/> [11 August 2007].

- Craig Lee and Domenico Talia. Grid Programming Models: Current Tools, Issues, and Directions. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 21. John Wiley and Sons, 2003.
- Craig Lee, Satoshi Matsuoka, Domenico Talia, Alan Sussmann, Matthias Mueller, Gabrielle Allen, and Joel Saltz. A Grid Programming Primer, August 2001. <http://www.cct.lsu.edu/~gallen/Reports/GridProgrammingPrimer.pdf> [7 February 2008].
- LHC. The Large Hadron Collider, 2007. <http://lhc.web.cern.ch/lhc/> [11 August 2007].
- LHC Communication. LHC Computing, 2007. <http://www.interactions.org/LHC/computing/index.html> [11 August 2007].
- Maozhen Li and Mark Baker. A Review of Grid Portal Technology. In Jose Cunha and Omer Rana, editors, *Grid Computing: Software Environments and Tools*, chapter 6, pages 126–156. Springer, 2006.
- Panos Livadas and David Small. Understanding Code Containing Preprocessor Constructs. In *WPC 1994: Proceedings of the Third Workshop on Program Comprehension*, November 1994.
- LSF. Platform Load Sharing Facility, 2008. <http://www.platform.com/Products/platform-lsf> [17 November 2008].
- Bertram Ludascher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, August 2006.
- MacroMates. TextMate - The Missing Editor for Mac OS X, 2008. <http://macromates.com/> [30 June 2008].
- Make. GNU make, 2006. http://www.gnu.org/software/make/manual/html_node/index.html [30 June 2008].

- Cameron Marlow, Mor Naaman, Danah Boyd, and Marc Davis. HT06, Tagging Paper, Taxonomy, Flickr, Academic Article, To Read. In *Hypertext 2006: Proceedings of the Seventeenth ACM Conference on Hypertext and Hypermedia*, August 2006.
- Kevin Marshall, Chad Pytel, and Jon Yurek. *Pro Active Record: Databases with Ruby and Rails*. Apress, 2007.
- Anthony Mayer, Steve McGough, Nathalie Furmento, Jeremy Cohen, Mur-taza Gulamali, Laurie Young, Ali Afzal, Steven Newhouse, and John Darlington. ICENI: An Integrated Grid Middleware to Support e-Science. In *Proceedings of the Workshop on Component Models and Systems for Grid Applications in conjunction with ICS 2004*, June 2004.
- Roger Menday. The Web Services Architecture and the UNICORE Gateway. In *ICIW 2006: Proceedings of the International Conference on Internet and Web Applications and Services*, February 2006.
- Roger Menday and Philipp Wieder. GRIP: The Evolution of UNICORE towards a Service Oriented Grid. In *CGW 2003: Proceedings of the 3rd Cracow Grid Workshop*, October 2003.
- Paul Messina. Challenges of the LHC: the computing challenge. *The European Physical Journal C - Particles and Fields*, 34(1), May 2004.
- Bertrand Meyer. Design by Contract. In Dino Mandrioli and Bertrand Meyer, editors, *Advances in Object-Oriented Software Engineering*, pages 1–50. Prentice Hall, 1991.
- MPI Forum. MPI-2: Extensions to the Message-Passing Interface, July 1997. <http://www.mpi-forum.org/docs/mpi-20.ps> [4 February 2008].
- MSG. Monash Sun Grid, 2008. <http://http://www.monash.edu.au/eresearch/services/mcg/msg.html> [17 November 2008].
- Matthias Muller, Matthias Hess, and Edgar Gabriel. Grid Enabled MPI Solutions for Clusters. In *CCGrid 2003: Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid*, May 2003.

- myExperiment. myExperiment Home, 2009. <http://www.myexperiment.org/> [4 February 2009].
- MySQL. MySQL Database, 2008. <http://www.mysql.com/> [30 May 2008].
- Hidemoto Nakada, Yoshio Tanaka, Satoshi Matsuoka, and Satoshi Sekiguchi. Ninf-G: A GridRPC System on the Globus Toolkit. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 25. John Wiley and Sons, 2003.
- NeSC. National e-Science Centre definition of e-Science, 2007. <http://www.nesc.ac.uk/nesc/define.html> [15 August 2007].
- NetBeans. Introduction to the NetBeans Platform, 2007a. <http://www.netbeans.org/products/platform/> [27 August 2007].
- NetBeans. Introduction to the NetBeans IDE, 2007b. <http://www.netbeans.org/products/ide/> [27 August 2007].
- Steven Newhouse, Jennifer M. Schopf, Andrew Richards, and Malcolm Atkinson. Study of User Priorities for e-Infrastructure for e-Research (SUPER), April 2007. http://www.nesc.ac.uk/technical_papers/UKeS-2007-01.pdf [15 August 2007].
- NGS. National Grid Service, 2007. <http://www.grid-support.ac.uk/> [10 September 2007].
- NorduGrid. NorduGrid: Grid Solution for Wide Area Computing and Data Handling, 2007. <http://www.nordugrid.org/> [10 September 2007].
- Ronald J. Norman and Jay F. Nunamaker, Jr. CASE Productivity: Perceptions of Software Engineering Professionals. *Communications of the ACM*, 32(9), September 1989.
- Jason Novotny, Michael Russell, and Oliver Wehrens. GridSphere: A Portal Framework for Building Collaborations. *Concurrency and Computation: Practice and Experience*, 16(5):503–513, March 2004.

- NSPR. Netscape Portable Runtime, 2008. <http://www.mozilla.org/projects/nspr/> [11 March 2008].
- OASIS. Web Services Resource 1.2 (WS-Resource), 2006a. http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf [30 January 2008].
- OASIS. Web Services Resource Properties 1.2 (WS-ResourceProperties), 2006b. http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf [30 January 2008].
- OASIS. Web Services Resource Lifetime 1.2 (WS-ResourceLifetime), 2006c. http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf [30 January 2008].
- OASIS. Web Services Service Group 1.2 (WS-ServiceGroup), 2006d. http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf [30 January 2008].
- OASIS. Web Services Base Faults 1.2 (WS-BaseFaults), 2006e. http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf [30 January 2008].
- OASIS. OASIS Standards and Other Approved Work, 2008a. <http://www.oasis-open.org/specs/index.php> [25 January 2008].
- OASIS. OASIS Web Services Resource Framework (WSRF), 2008b. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf [30 January 2008].
- OGCE. The Open Grid Computing Environments Portal and Gateway Toolkit, 2008. http://www.collab-ogce.org/ogce/index.php/Main_Page [18 February 2008].
- Tom Oinn, Mark Greenwood, Matthew Addis, M. Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, Peter Li, Phillip Lord, Matthew R. Pocock, Martin Senger,

- Robert Stevens, Anil Wipat, and Chris Wroe. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, August 2006.
- John Ousterhout. Scripting: Higher Level Programming for the 21st Century. *IEEE Computer*, 31(3), March 1998.
- Manish Parashar and James C. Browne. Conceptual and Implementation Models for the Grid. *Proceedings of the IEEE*, 93(3), March 2005.
- PBS. PBS GridWorks, 2008. <http://www.pbsgridworks.com/> [17 November 2008].
- Tom Peachey, David Abramson, Andrew Lewis, Donny Kurniawan, and Rhys Jones. Optimization using Nimrod/O and its Application to Robust Mechanical Design. In *PPAM 2003: Proceedings of the Fifth International Conference on Parallel Processing and Applied Mathematics*, September 2003.
- Perl. The Perl Directory - perl.org, 2008. <http://www.perl.org/> [30 June 2008].
- Remko Popma. JET Tutorial Part 1 (Introduction to JET), May 2004a. http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html [5 February 2008].
- Remko Popma. JET Tutorial Part 2 (Write Code that Writes Code), May 2004b. http://www.eclipse.org/articles/Article-JET2/jet_tutorial2.html [5 February 2008].
- PostgreSQL. PostgreSQL Database, 2008. <http://www.postgresql.org/> [30 May 2008].
- PRAGMA. Pacific Rim Application and Grid Middleware Assembly, 2008. <http://www.pragma-grid.net/> [17 November 2008].

Lutz Prechelt. Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java. *Advances in Computers*, 57, June 2003.

Ptolemy II. Ptolemy II Project - Heterogenous Modelling and Design, 2008. <http://ptolemy.eecs.berkeley.edu/ptolemyII/> [11 February 2008].

Python. Python Programming Language - Official Website, 2008. <http://www.python.org/> [30 June 2008].

Qt. Qt - Trolltech, 2008. <http://trolltech.com/products/qt/> [22 January 2008].

Rails. Ruby on Rails, 2008. <http://www.rubyonrails.org/> [26 May 2008].

Rake. Rake - Ruby Make, 2006. <http://rake.rubyforge.org/> [30 June 2008].

Michael Rambadt, Rebecca Breu, Luca Clementi, Thomas Fieseler, Andre Giesler, Wolfgang Gurich, Paolo Malfetti, Roger Menday, Johannes Reetz, and Achim Streit. DEISA and D-Grid: using UNICORE in production Grid infrastructures. In *GES 2007: Proceedings of the German e-Science Conference*, May 2007.

Charles Reis and Robert Cartwright. Taming a professional IDE for the classroom. In *SIGCSE 2004: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, March 2004.

Dirk Riehle. The Event Notification Pattern - Integrating Implicit Invocation with Object-Orientation. *Theory and Practice of Object Systems*, 2(1), 1996.

Rocks-52. Rocks-52 Cluster, 2008. <http://rocks-52.sdsc.edu/wordpress/> [17 November 2008].

Mathilde Romberg. The UNICORE Architecture: Seamless Access to Distributed Resources. In *HPDC 1999: Proceedings of the 8th International Symposium on High Performance Distributed Computing*, August 1999.

- Mathilde Romberg. The UNICORE Grid Infrastructure. *Scientific Programming*, 10(2), 2002.
- David De Roure, Mark A. Baker, Nicholas R. Jennings, and Nigel R. Shadbolt. The Evolution of the Grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 3. John Wiley and Sons, 2003.
- Winston W. Royce. Managing the Development of Large Software Systems: Concepts and Techniques. In *Proceedings of the IEEE WESCON Conference*, August 1970.
- RSCM. Ruby Source Control Management, 2006. <http://rscm.rubyforge.org/> [12 June 2008].
- Ruby. Ruby Programming Language, 2008. <http://www.ruby-lang.org/> [26 May 2008].
- Michael Russell, Jason Novotny, and Oliver Wehrens. The Grid Portlets Web Application: A Grid Portal Framework. In *GAMW 2005: Proceedings of the Second Workshop on Grid Application and Middleware in conjunction with PPAM 2005*, September 2005.
- Aaron Rustad. Ruby on Rails and J2EE: Is There Room for Both?, July 2005. <http://www-128.ibm.com/developerworks/linux/library/wa-rubyonrails/?ca=dgr-lnxw16RubyAndJ2EE> [26 May 2008].
- SAGA-RG. The Simple API For Grid Applications Research Group (SAGA-RG), 2008. <https://forge.gridforum.org/projects/saga-rg/> [20 February 2008].
- Mitsuhisa Sato, Taisuke Boku, and Daisuke Takahashi. OmniRPC: A Grid RPC System for Parallel Programming in Cluster and Grid Environment. In *CCGrid 2003: Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid*, May 2003.
- Michael Scott. *Programming Language Pragmatics*. Morgan Kaufmann, second edition, 2006.

- SDSS. The Sloan Digital Sky Survey, 2007. <http://www.sdss.org/> [11 August 2007].
- Simon See, Jie Song, Liang Peng, Appie Stoelwinder, and Hoon Kang Neo. GridE: A Grid-Enabled Development Environment. In *GCCW 2003: Proceedings of the Second International Conference on Grid and Cooperative Computing Workshops*, December 2003.
- Ahmed Seffah and Juergen Rilling. Investigating the Relationship between Usability and Conceptual Gaps for Human-Centric CASE Tools. In *HCC 2001: Proceedings of the IEEE International Symposium on Human-Centric Computing Languages and Environments*, September 2001.
- Keith Seymour, Hidemoto Nakada, Satoshi Matsuoka, Jack Dongarra, Craig Lee, and Henri Casanova. Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In *Grid 2002: Proceedings of the Third International Workshop on Grid Computing*, November 2002.
- SGE. Sun Grid Engine, 2008. <http://www.sun.com/software/gridware/> [17 November 2008].
- David Snelling. UNICORE and the Open Grid Services Architecture. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 29. John Wiley and Sons, 2003.
- Harold Soh, Shazia Haque, Weili Liao, and Rajkumar Buyya. Grid Programming Models and Environments. In Yuan-Shun Dai, Yi Pan, and Rajeev Raje, editors, *Advanced Parallel and Distributed Computing: Evaluation, Improvement, and Practice*, chapter 8. Nova Science, 2006.
- Ian Sommerville. *Software Engineering*. Addison-Wesley, eight edition, 2006.
- Borja Sotomayor and Lisa Childers. *Globus Toolkit 4: Programming Java Services*. Morgan Kaufmann, December 2005.
- Daniel Spooner, Junwei Cao, Stephen Jarvis, Ligang He, and Graham Nudd. Performance-Aware Workflow Management for Grid Computing. *The Computer Journal*, 48(3), 2005.

- SQLite. SQLite Database, 2008. <http://www.sqlite.org/> [30 May 2008].
- Heinz Stockinger. Defining the grid: a snapshot on the current view. *The Journal of Supercomputing*, 42(1), October 2007.
- Achim Streit, Dietmar Erwin, Thomas Lippert, Daniel Mallmann, Roger Menday, Michael Rambadt, Morris Riedel, Mathilde Romberg, Bernd Schuller, and Philipp Wieder. UNICORE - From Project Results to Production Grids. In Lucio Grandinetti, editor, *Grid Computing: The New Frontier of High Performance Computing*. Elsevier, November 2005.
- Mathias Stumpert. g-Eclipse Architecture, 2007. <http://www.eclipse.org/geclipse/resources/D1.5.pdf> [11 January 2008].
- Subversion. The Subversion project at Tigris.org, 2007. <http://subversion.tigris.org/> [28 August 2007].
- Joe Szurszewski. We Have Lift-off: The Launching Framework in Eclipse, 2003. <http://www.eclipse.org/articles/Article-Launch-Framework/launch.html> [16 June 2008].
- Yoshio Tanaka, Hidemoto Nakada, Satoshi Sekiguchi, Toyotaro Suzumura, and Satoshi Matsuoka. Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing*, 1(1), March 2003.
- Bruce Tate. *From Java to Ruby: Things Every Manager Should Know*. Pragmatic Bookshelf, 2006.
- TAU. TAU - Tuning and Analysis Utilities, 2008. <http://www.cs.uoregon.edu/research/tau/home.php> [30 June 2008].
- Mary Thomas and John Boisseau. Building Grid Computing Portals: the NPACI Grid Portal Toolkit. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 28. John Wiley and Sons, 2003.

Mary Thomas, Steve Mock, Maytal Dahan, Kurt Mueller, Don Sutton, and John Boisseau. The GridPort Toolkit: A System for Building Grid Portals. In *HPDC 2001: Proceedings of the 10th International Symposium on High Performance Distributed Computing*, 2001.

Michio Tsuda, Yosuke Morioka, Masato Takadachi, and Mayumi Takahashi. Productivity analysis of software development with an integrated CASE tool. In *ICSE 1992: Proceedings of the 14th International Conference on Software Engineering*, June 1992.

UNICORE. UNICORE - Distributed Computing and Data Resources, 2008. <http://www.unicore.eu/> [31 January 2008].

Valgrind. Valgrind Home, 2008. <http://valgrind.org/> [30 June 2008].

Dimitri van Heesch. Doxygen, 2008. <http://www.stack.nl/~dimitri/doxygen/> [30 June 2008].

Rob van Nieuwpoort, Jason Maassen, Gosia Wrzesiska, Rutger Hofman, Cerial Jacobs, Thilo Kielmann, and Henri Bal. Ibis: A Flexible and Efficient Java-Based Grid Programming Environment. *Concurrency and Computation: Practice and Experience*, 17:1079–1107, 2005.

VDT. The Virtual Data Toolkit, 2007. <http://vdt.cs.wisc.edu/> [14 September 2007].

Vim. Vim online, 2008. <http://www.vim.org/> [30 June 2008].

Visual Studio. Visual Studio Developer Center, 2008. <http://msdn.microsoft.com/vstudio/> [22 January 2008].

Gregor von Laszewski and Kaizar Amin. Grid Middleware. In Qusay Mahmoud, editor, *Middleware for Communications*, chapter 5. John Wiley and Sons, 2004.

Gregor von Laszewski, Jarek Gawor, Sriram Krishnan, and Keith Jackson. Commodity Grid Kits - Middleware for Building Grid Computing Environments. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid*

Computing: Making the Global Infrastructure a Reality, chapter 26. John Wiley and Sons, 2003.

VSIP. Visual Studio Industry Partner, 2008. <http://msdn.microsoft.com/vsip/> [22 January 2008].

VSTS. Visual Studio Team System, 2008. <http://msdn.microsoft.com/teamssystem/> [22 January 2008].

W3C. Web Services Architecture - W3C Working Group Note 11 February 2004, 2004. <http://www.w3.org/TR/ws-arch/> [25 January 2008].

W3C. Extensible Markup Language (XML) 1.0 (Fourth Edition) - W3C Recommendation 16 August 2006, 2006. <http://www.w3.org/TR/xml/> [25 January 2008].

W3C. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) - W3C Recommendation 27 April 2007, 2007a. <http://www.w3.org/TR/soap12-part1/> [25 January 2008].

W3C. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language - W3C Recommendation 26 June 2007, 2007b. <http://www.w3.org/TR/wsd120/> [25 January 2008].

W3C. Web Services Activity, 2008. <http://www.w3.org/2002/ws/> [25 January 2008].

Anthony Wasserman. Tool Integration in Software Engineering Environments. In *Proceedings of the International Workshop on Environments (Software Engineering Environments)*, September 1989.

Greg Watson. *The Design and Implementation of a Parallel Relative Debugger*. PhD thesis, Monash University, 2000.

David Watt and Deryck Brown. *Programming Language Processors in Java: Compilers and Interpreters*. Prentice Hall, 2000.

- WEBrick. WEBrick - an HTTP Server Toolkit, 2008. <http://www.webrick.org/> [3 June 2008].
- Roy Williams. Grids and the Virtual Observatory. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, chapter 38. John Wiley and Sons, 2003.
- Dave Winer. XML-RPC Specification, 1999. <http://www.xmlrpc.com/spec/> [25 January 2008].
- WSRF. The WS-Resource Framework, 2008. <http://www.globus.org/wsrp/> [30 January 2008].
- Xcode. Apple Developer Connection - Xcode, 2008. <http://developer.apple.com/tools/xcode/> [22 January 2008].
- Helen Xiang, Mark Baker, and Robert Nichol. Experiences Mirroring and Distributing the Sloan Digital Sky Survey. In *GCCW 2006: Proceedings of the Fifth International Conference on Grid and Cooperative Computing Workshops*, October 2006.
- Xiaoyu Yang, Martin Dove, Mark Hayes, Mark Calleja, Ligang He, and Peter Murray-Rust. Survey of Major Tools and Technologies for Grid-enabled Portal Development, September 2006. <http://www.allhands.org.uk/2006/proceedings/papers/622.pdf> [18 February 2008].
- Zhihui Yang and Michael Jiang. Using Eclipse as a Tool-Integration Platform for Software Development. *IEEE Software*, 24(2), 2007.
- Asim YarKhan, Jack Dongarra, and Keith Seymour. GridSolve: The Evolution of A Network Enabled Solver. In *WoCo9: Proceedings of the IFIP Working Conference on Grid-based Problem Solving Environments*, July 2006a.
- Asim YarKhan, Keith Seymour, Kiran Sagi, Zhiao Shi, and Jack Dongarra. Recent Developments in GridSolve. *International Journal of High Performance Computing Applications*, 20(1), 2006b.

- Jia Yu and Rajkumar Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3:171–200, 2006.
- Andreas Zeller. The Future of Programming Environments: Integration, Synergy, and Assistance. In *FOSE 2007: Future of Software Engineering 2007*, May 2007.
- Chongjie Zhang, Ian Kelley, and Gabrielle Allen. Grid Portal Solutions: A Comparison of GridPortlets and OGCE. *Concurrency and Computation: Practice and Experience*, 19(12):1739–1748, August 2007.

Colophon

Colophon : a statement at the end of the book, typically with a printer's emblem, giving information about its authorship and printing.

Rather than talking about the printing process of this thesis, I would like to use the next few pages to personally thank a number of people who have motivated, helped, and encouraged me. They may not realize their support but I am grateful and I am blessed by them and their friendship.

- Glen Tarmidi, Christian Tirta, Petrus Diredja, Myke, Pasryani, Intan.
- Stephen Gunawan, Yuliana, Soendjojo Logianto.
- July Teh, Vera Natalea.
- K'Sendjaya, O'Hadi, Benidictus Jobeanto, Josep Widjaja, Felicia.
- C'Lyfie Sugianto, O'Willy, T'Diana.
- Mona Soetanto, Cordelia Selomulya, Fenny Wiradjaja, Meita.
- Toh Yen Pang, Toh Sin Yow, Dorothy Chan.
- Lydia Teh, Doen Ming Ong.
- Sheik Yan Wong, Wan Ling Chua, Yan Yan Chiong, Zhi Kai Chua.
- Jason Lau, Barbara Wee.
- En Ye Ong, Sean Tan.
- Jonathan Chau, Fernando Tan, Renhaw Lim.