

ISENGARD: an infrastructure for supporting e-science and grid application development

Donny Kurniawan^{*,†} and David Abramson

Caulfield School of Information Technology, Monash University, Vic. 3145, Australia

SUMMARY

Grid computing facilitates the aggregation and coordination of resources that are distributed across multiple administrative domains for large-scale and complex e-Science experiments. Writing, deploying, and testing grid applications over highly heterogeneous and distributed resources are complex and challenging. The process requires grid-enabled programming tools that can handle the complexity and scale of the infrastructure. However, 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. This paper presents the design and implementation of ISENGARD, an infrastructure for supporting e-Science and grid application development. ISENGARD provides services, tools, and APIs that simplify grid software development. Copyright © 2010 John Wiley & Sons, Ltd.

Received 10 November 2009; Revised 4 July 2010; Accepted 8 July 2010

KEY WORDS: grid computing; IDE; application development

1. INTRODUCTION

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 [1, 2]. 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 [3]. 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. However, no common protocols and standards have been defined in the area of grid application development [4]. 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 [5].

The aim of this paper 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

^{*}Correspondence to: Donny Kurniawan, Caulfield School of Information Technology, Monash University, Vic. 3145, Australia.

[†]E-mail: donny.kurniawan@infotech.monash.edu.au

of grid programmers and e-Scientists. ISENGARD consists of three distinct software components with well-defined APIs and bindings [4–7]. These components include user-level applications, high-level tools, and low-level middleware services.

1.1. Motivation and objectives

Grid middleware hides much of the complexity of the underlying resources and network fabric. Utilizing grid resources requires appropriate software that needs to be specifically written to take advantage of the infrastructure. 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 [8]:

‘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 [9]. There are several specific challenges that make developing grid applications non-trivial and they 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 situation may also be exacerbated if a programmer wishes to use resources that 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 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 [8].

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. 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 [8].

2. ARCHITECTURE AND DESIGN

To address the aforementioned issues introduced in the preceding section, we have developed a new infrastructure that supports the development of e-Science and grid applications (ISENGARD). The infrastructure 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) [6, 7], Worqbench [4], and GridDebug [5]. RDT is an IDE framework that 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 1, which also depicts the level of abstraction. GridDebug and Worqbench can be categorized as lower and upper middleware layers, respectively, and RDT 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.

As introduced in Section 1, grid middleware provides the necessary infrastructure such as security and access management. Rather than duplicating these underlying functions, ISENGARD is designed on top of the middleware and it fully relies on the respective middleware for resource access and security infrastructure. Thus, users need to provide ISENGARD with typical authentication information, for example, X.509 proxy certificates. More importantly, ISENGARD does not place any additional security requirements or restrictions such as open firewall ports beyond those required by the middleware.

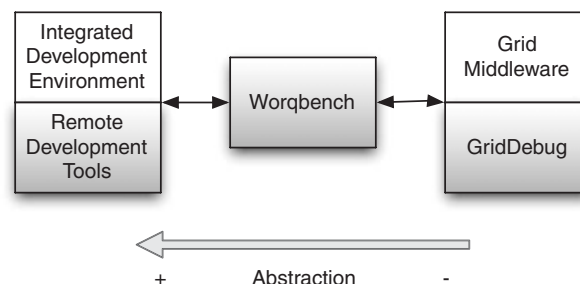


Figure 1. Architecture of ISENGARD.

The next three sections describe the architecture and design of GridDebug (Section 2.1), Worqbench (Section 2.2), and RDT (Section 2.3) in detail.

2.1. GridDebug framework

GridDebug 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 [10, 11], meaning that the 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 [12], 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.

GridDebug also provides a 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 as a solution to address issues such as job scheduling and access restrictions across a hierarchy of resources.

2.1.1. 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 a debug client, an interface to grid middleware, a debug library, and a debug back-end wrapper. Figure 2 shows the components and the overall architecture in relation to the grid middleware.

- *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 [13] or Eclipse [14]. Furthermore, developers can extend or write sophisticated debug clients, for example, high-level tracers and tools for enforcing software contracts (pre and post conditions) [15].

- *Middleware Compatibility Layer.* GridDebug encapsulates a *generic* debug library as a *middleware-specific* extension. This encapsulation is provided by the middleware compati-

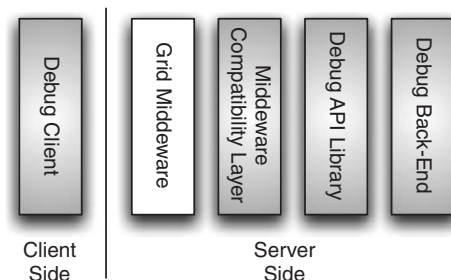


Figure 2. GridDebug architecture.

bility 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 the OGSA-based middleware, the compatibility layer is implemented as a grid service that needs to be deployed into a service container.

- *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 2.1.2.

- *Debug Back-End.* GridDebug leverages well-proven traditional debuggers as its debug back-end. These debuggers, for example GDB [12], 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 [16], can be utilized by implementing an interface to the debug library, without changing the API method signatures.

2.1.2. GridDebug API. 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 [17]. 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. The components of the debug API library are depicted conceptually in Figure 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 interdependent components [18]. 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 [12] to be utilized as the back-end debug engine. Section 2.1.3 discusses the GridDebug remote debugging mechanism in greater detail.

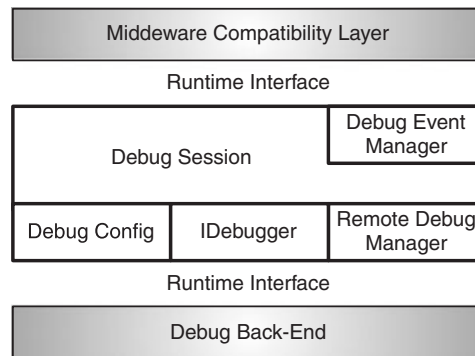


Figure 3. Components of the debug API library.

- *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.

2.1.3. Grid remote debugging. In a typical grid environment, program execution is managed by a local queue manager (for example, PBS [19], LSF [20], and SGE [21]), 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 a callback mechanism for grid remote debugging. GridDebug employs a small auxiliary program whose sole purpose is to contact the debug back-end; and act as a transparent proxy for input and output streams of the debugged application. In this mechanism, it is the debugger that waits for a callback connection from the debugged application. This mechanism 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 this remote debugging mechanism is depicted in Figure 4.

2.2. 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. 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.

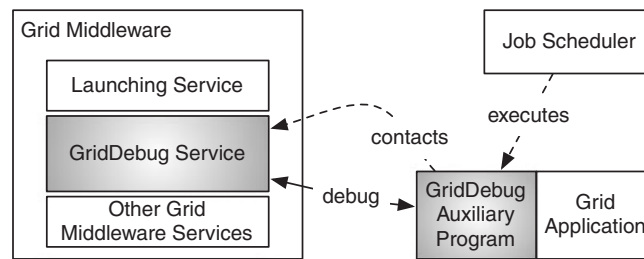


Figure 4. Grid remote debugging.

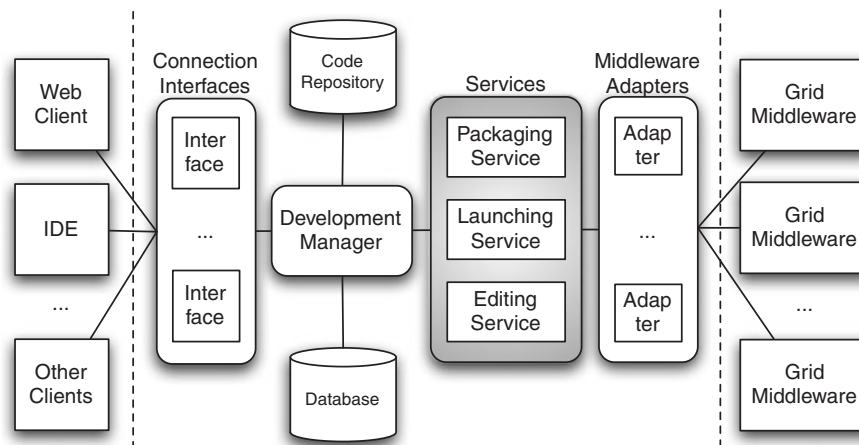


Figure 5. Key components in Worqbench.

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 1. 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 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 user interactions with Worqbench. For example, a web browser with a Java applet or an Ajax user interface library [22] can be used to access Worqbench. This provides an accessible and ubiquitous platform similar to a grid portal for developing grid software. Alternatively, a program with a CLI 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.

2.2.1. Worqbench components. 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. These components are depicted in Figure 5.

- **Connection Interface.** The 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 [23] or XML-RPC [24]. The connection interface provides a clean and well-defined abstraction layer that allows Worqbench to support various IDEs and clients.
- **Code Repository.** Worqbench functions, for example, compiling and deploying an application on grid resources, require access to the application's project. Worqbench employs a code

repository to manage and store the contents and metadata details of application projects. 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.

- *Development Manager*. The 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 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 2.2.2.
- *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.
- *Development Services*. Worqbench services offer 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 and required libraries to remote machines; setting up and initializing the appropriate directories; and executing a build script to compile the application. The service requires the target resources to have development tools such as compilers and linkers 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.

- *Middleware Adapter*. The middleware adapter 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.

2.2.2. 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 6. The diagram uses Barker’s Notation [25] to represent relationships between entities with connecting lines and symbols that indicate the cardinality of the relationship.

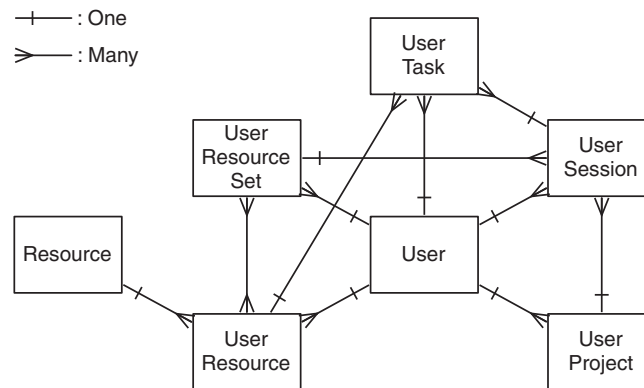


Figure 6. System model of Worqbench.

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 6 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.
- *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 and source code 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 to 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 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.

2.3. Remote development tools

RDT are a set of modules and a user-level framework that augment an IDE for grid application development. While 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. 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 [13] or Eclipse [14].

The architecture of RDT is depicted in Figure 7. 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 [26]. The user interface layer can be customized according to the base IDE while retaining the principal non-UI part.

2.3.1. RDT framework. IDEs are mainly concerned with software projects, folders, and files as opposed to target resources. Thus, the process of developing, executing, and debugging grid applications over a *large number of heterogeneous resources* becomes cumbersome. To alleviate this problem, RDT introduces several improvements.

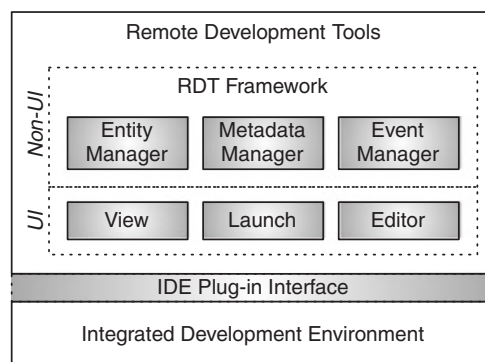


Figure 7. RDT architecture overview.

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.

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. 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.

While these ideas are valuable even for conventional software development, we believe that 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.

- *Entity Manager*. The RDT framework maintains an extended IDE model that is based on the Worqbench design (Section 2.2.2) since it is comprehensive and adequate to represent common grid application development scenarios in the IDE. The administration of RDT entities are handled by various entity managers. Managers provide RDT users with *single access points* that simplify the management and coordination of entities across the system. They communicate with various Worqbench services and act as translators between Worqbench and RDT system models.
- *Metadata Manager*. The 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 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 [27]. 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. 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.
- *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 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.

- *RDT API*. 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 that 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.

2.3.2. 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 is inadequate in the grid where multiple machines may be involved in the development process.

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 with 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. SYSTEM IMPLEMENTATION

This section presents the prototype implementation of ISENGARD. We describe the implementation details of the GridDebug framework in Section 3.1, followed by the details of Worqbench in Section 3.2 and RDT in Section 3.3.

3.1. GridDebug implementation

We have developed and implemented GridDebug using a number of popular software toolkits: WSRF [28, 29], Globus Toolkit 4 [30–32], and GDB/GDBServer [12]. 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 8 gives an overview of GridDebug components and the implementation. 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. The implementation of GridDebug is described in greater detail in the following three sections.

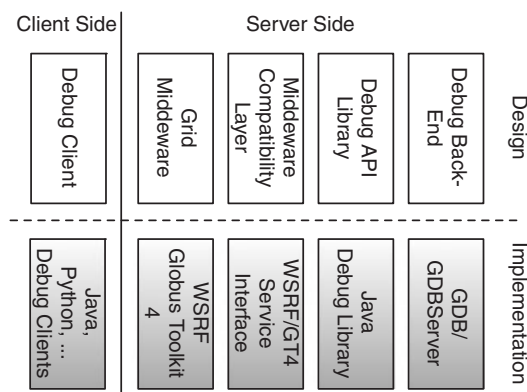


Figure 8. GridDebug implementation overview.

3.1.1. 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 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 while 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.

The GridDebug debugging service has been implemented using a design pattern known as the factory/instance pattern [32]. 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; and they are described 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 2.1.2) 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 relationship between the components is depicted in Figure 9 and the interaction between the client and the GridDebug service can be outlined as follows:

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.

3.1.2. 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

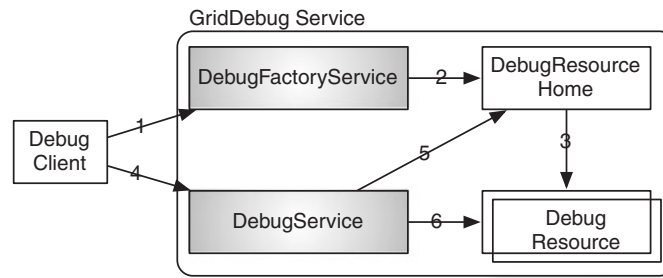


Figure 9. Components of GridDebug service implementation.



Figure 10. Class diagram for GDBDebugger.

library is implemented in Java and it consists of five Java classes (Figure 3). *DebugSession* 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 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.

The *IDebugger* specifies a set of GridDebug APIs based on the High-Performance Debugging Forum (HPDF) standard [33, 34]. The HPDF standard is the result of significant research on an appropriate command set for a 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. The methods follow the specification as set by the HPDF standard and they are named according to their HPDF command counterparts. 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 [33]. 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.

3.1.3. GridDebug Back-End. The debug back-end is the software component that performs the actual debugging operations as defined by the GridDebug API. We have developed a back-end debugger for GridDebug by utilizing GDB and GDBServer [12] with a small alteration. The necessary modification improves the debugger's capability to handle situations where a job scheduler is employed to execute applications. It allows GDBServer to *initiate the remote debugging connection* when it is ready to debug an application.

The GridDebug back-end is implemented by the *GDBDebugger* class that extends the *AbstractDebugger* class, which in turn implements the *IDebugger* interface class. The class diagram is shown in Figure 10. 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.

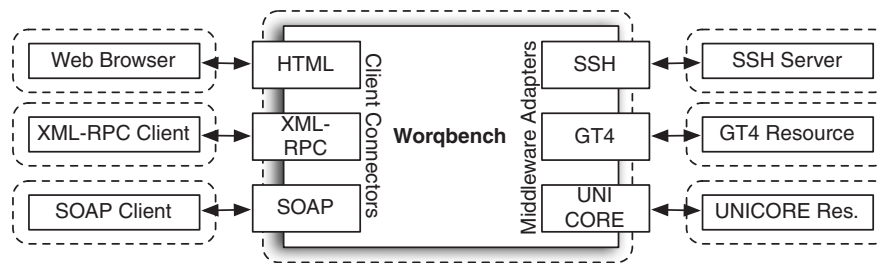


Figure 11. Client connectors and middleware adapters.

3.2. Worqbench implementation

As discussed in Section 2.2, Worqbench provides a framework that offers an API and high-level services that bridge between IDEs and grid middleware. It allows developers to utilize a set of grid resources where each resource is managed by different grid middlewares. Worqbench is written in Ruby [35] 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 pattern [36]. 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 [37]. In addition, Rails offers several features that serve as a sound starting point [38, 39]. The Worqbench implementation is discussed in greater detail in the following sections.

3.2.1. Client connectors. We have implemented three client connectors for HTML pages, XML-RPC, and SOAP protocols (Figure 11). 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. 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. The network connection can be encrypted by running Worqbench with a web server that supports HTTPS with Secure Sockets Layer or Transport Layer Security protocol [40].

3.2.2. 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 11). 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 12 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.

3.2.3. Worqbench data structures and APIs. Worqbench defines several Rails model classes to implement the system entities (Section 2.2.2). Figure 13 shows the model classes and their public data members. It is an implementation-oriented diagram of Figure 6 in Section 2.2.2. The classes have internal hashes that store attributes in the form of key/value pairs. API methods, namely

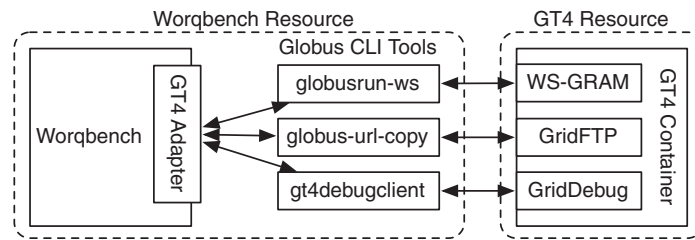


Figure 12. Interaction between Worqbench, CLI tools, and GT4 services.

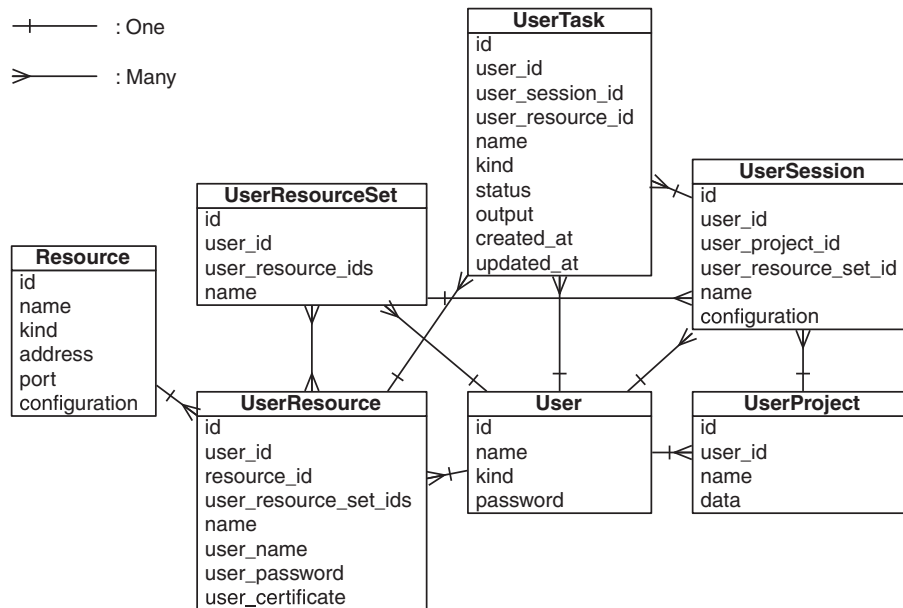


Figure 13. Worqbench model classes.

SetAttr() and GetAttr(), can be used to assign and retrieve user-defined custom attributes. All Worqbench API methods, with the exception of admin.GetKey(), require a session key as a parameter. 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.

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.

3.2.4. Worqbench services. Worqbench implements grid development functions as web services that are invoked through remote procedure calls. The API methods operate on a UserSession object with its application project and resource set. Worqbench provides methods that correspond to the engineering stages in a software development life cycle: Packaging Service, Launching Service, and Editing Service. The API methods are listed in Table I.

The Packaging Service provides methods for transferring and compiling an application project on a set of remote resources. In the current implementation, a user specifies the packaging/build operation by writing a shell script. The script is then executed by the Worqbench Packaging Service by passing it to a command-line interpreter on the remote resources. 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 Editing Service offers a set

Table I. Worqbench services API methods.

Packaging service	Launching service	Editing service
<code>packaging.Transfer()</code>	<code>launching.Run()</code>	<code>editing.Checkout()</code>
<code>packaging.Init()</code>	<code>launching.Debug()</code>	<code>editing.Commit()</code>
<code>packaging.Build()</code>	<code>launching.Command()</code>	<code>editing.GetRevisions()</code>
<code>packaging.Clean()</code>		<code>editing.IsUpdated()</code>

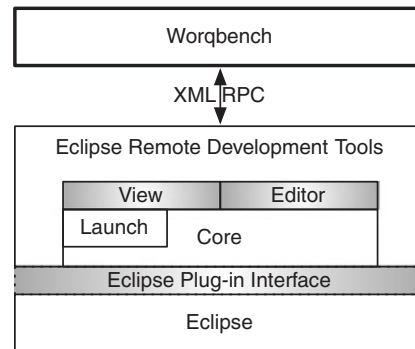


Figure 14. Eclipse RDT and Worqbench.

of API methods for managing a software project including its multiple revisions on a user's computer.

3.3. RDT implementation

As introduced in Section 2.3, RDT augments the IDE through its plug-in extension mechanism and leverages the GridDebug framework and Worqbench services for grid application development. We have developed an implementation of RDT for Eclipse called Eclipse RDT. It is implemented as several plug-ins written in Java that can be loaded into the IDE. For the automation facility, the Eclipse RDT features an embedded interpreter for the Groovy programming language [41]. 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, and closures. 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.

3.3.1. IDE integration. Eclipse RDT is implemented according to the Eclipse plug-in guidelines [42]. It is divided into two software layers, UI and non-UI; and it consists of several Java packages. Eclipse RDT communicates with Worqbench using the XML-RPC client connector as illustrated in Figure 14. RDT invokes synchronous XML-RPC calls to access and utilize Worqbench services and entities.

Eclipse defines a number of models such as `IFile`, `IFolder`, and `IProject` which provide abstractions for local files, folders, and projects, respectively [43]. 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. 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.

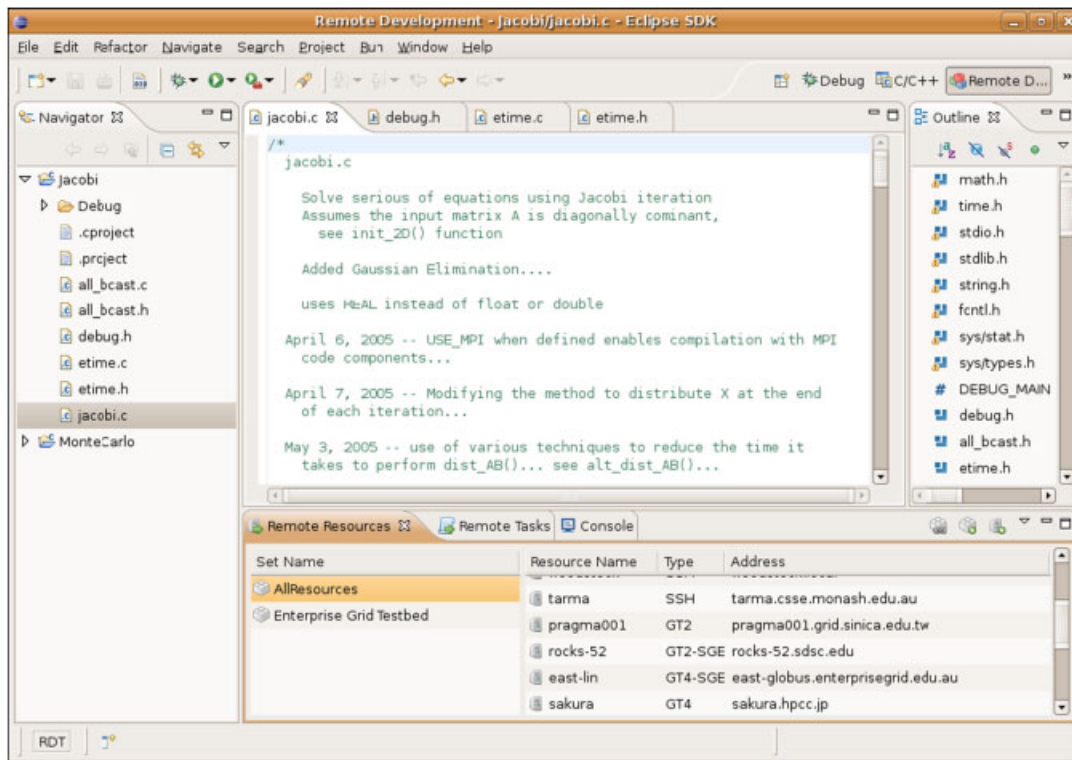


Figure 15. Eclipse RDT and its Remote Development perspective.

Eclipse has the concept of a launch configuration, which is defined as a description of how to run or debug a program [44]. 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.

3.3.2. User interface. Eclipse RDT has its own Eclipse perspective, the Remote Development perspective, which 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. Figure 15 shows the Eclipse RDT and its Remote Development perspective. We have also implemented an Eclipse preferences dialog for configuring RDT settings and an Eclipse launch configuration dialog for launching remote grid applications.

3.3.3. 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 3.2.3) 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.

3.3.4. 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 APIs are implemented as Java objects and methods that can be called and accessed directly from within Groovy scripts. In addition, built-in Eclipse APIs including the graphical user interface widget (e.g. dialog boxes) APIs are also *callable* within the 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.
- 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.

3.3.5. 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. 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. Built-in Eclipse APIs, data structures, and GUI widgets can be used within the Groovy scripts. Thus, the callback methods can display a pop-up dialog box, close Eclipse perspectives, invoke the compile action, run an executable, and so forth based on the occurrence of RDT events.

4. RELATED WORK

In this section, we outline the related work in the area of grid debugging tools and development environments. A comprehensive review of grid programming environments and tools deserves a paper in its own right. Interested readers are referred to [45–48]; whereas Goscinski [49] provides a survey of various grid deployment systems.

There are several debuggers that are designed for testing and debugging grid applications. Some examples of them are: Portable Parallel/Distributed Debugger (p2d2) [50], the metad debugger in the Harness framework [51], Net-dbx-G [52], the Mercury Monitoring System [53], and PDB [54].

The p2d2 is a project at the NASA Ames Research Center that developed a debugger for applications running on heterogeneous computational grids [50]. It employs a client–server architecture and relies on GDB [12] as the low-level portable debugger. Instances of GDB communicate with a debug server which is implemented in C++ and maintains a collection of C++ objects to represent processes and stacks. The server is controlled by a graphical user interface, the debug client, which allows users to examine the state and to control the execution of grid applications.

Harness is a metacomputing system that defines a simple but powerful architectural model to overcome the limited flexibility of traditional distributed software frameworks [51]. It consists of a kernel and plug-ins that provide various services for users. It is implemented in Java to leverage the homogenous architecture, the JVM, over heterogeneous computer platforms. Harness provides a distributed virtual machine for execution of metacomputing applications written in Java with Remote Method Invocations. A metad debugger has been developed in the Harness framework using the Java Platform Debug Architecture with remote debugging capability. The debugger is closely intertwined with the framework and it cannot be used with other grid middleware.

Net-dbx-G is a web-based debugger for Message Passing Interface (MPI) programs executing on grid resources [52]. It uses Java applets as the user interface and GDB [12] as the back-end debugger. It supports Globus Toolkit 2 (GT2) middleware with MPICH-G2 that provides the MPI programming library. Debugging an application requires the executable to be compiled with the Net-dbx-G instrumentation library. When the application is executed, an initialization method in the library notifies the debug client and spawns a child GDB process to debug the parent process. The user then controls GDB from a web browser using the Java applets.

The Mercury Monitoring System that is developed as part of the GridLab project is a generic grid monitoring framework [53]. It provides support mainly for application monitoring, however, the recent release of Mercury allows users to perform remote debugging. It employs GDB and GDBServer [12]. GDBServer is used as the debug server on grid nodes with GDB as the user interface on a programmer's desktop. Debugging is performed by sending a message to the Mercury monitoring library that is compiled into the application. The library then forks GDBServer and instructs it to attach to the debugged process. The developer controls the debug server using GDB from the desktop.

PDB is an implementation of the pervasive debugging approach [54]. It leverages the Xen Virtual Machine to virtualize the system resources used by a debugged application. The virtualization allows a user to control and to inspect the complete state of the application and the resources including their low-level details, such as processor instructions, system timers, and thread schedulers. By using Xen to virtualize grid resources, users can deterministically debug grid applications. However, this deterministic debugging technique is difficult to attain for applications that need to be run on a large-scale distributed system.

Different to all the debuggers described, GridDebug is designed as a debug library with high-level API suitable for debugging grid applications. It has a layered and modular architecture that allows it to be plugged into any grid middleware and to be used with any debug back-ends.

A number of projects have focused on providing a suitable development environment for grid applications. Indicative examples that represent the current state of research are: P-GRADE, g-Eclipse, GDT, and Introduce.

Parallel Grid Runtime and Application Development Environment (P-GRADE) [55] from MTA SZTAKI in Hungary provides a high-level graphical environment to develop parallel applications for parallel systems and grid. 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: Parallel Virtual Machine [56], MPI [57, 58], and Grid Application Toolkit [59].

The g-Eclipse [60, 61] 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. It supports several grid middleware such as gLite [62] and GRIA [63]. In addition, g-Eclipse can also be used to manage and access cloud computing resources such as Amazon Web Services [64].

Grid Development Tools (GDT) [65, 66] is a set of Eclipse IDE plug-ins for service-oriented grid application development. It is part of the Marburg *Ad hoc* Grid Environment, 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 [67–69] by dividing the development process into three separate model layers and automatically transforming models from one layer into the other.

Introduce [70] 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.

These grid IDEs 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 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.

5. CONCLUSION

It is difficult 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. Regardless of the difficulty, a formal usability study is certainly required to improve ISENGARD to better suit the needs of users and e-Scientists. We are preparing a usability study and a report as a follow up to this paper.

We have utilized GridDebug, Worqbench, and RDT in case studies that reflect common situations faced by grid software developers and users. The case studies are concerned with a small Jacobi method application and a substantial software package for molecular dynamics simulation, called GROMACS [71], consisting of approximately 300 000 lines of C code across 700 source files. To perform the case studies, we have developed a grid testbed that consists of various resources within several administrative domains. The resource details are given in Table II. In the case studies, we conducted various development activities such as editing and compiling source files; executing and debugging the compiled applications. Detailed treatment of the case studies including the testbed environment is given in [72]. The case studies demonstrate 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 application. The proof-of-concept implementations provide workable solutions to the problems and challenges identified in Section 1.

To conclude, we have designed and developed 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. It consists of a middleware service, a modular grid development framework, and a programmable IDE framework. We identify and present several significant outcomes and conclusions in this paper:

- *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 a 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

Table II. Testbed resources.

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

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 ‘off-line’ 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 does not 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 that 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.

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.

REFERENCES

1. Hey T, Trefethen A. e-Science and its implications. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 2003; **361**(1809):1809–1825.
2. NeSC. National e-Science Centre definition of e-Science, 2007. Available at: <http://www.nesc.ac.uk/nesc/define.html> [15 August 2007].
3. Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of Supercomputer Applications* 2001; **15**(3):200–222.

4. Kurniawan D, Abramson D. Worqbench: An integrated framework for e-Science application development. *e-Science 2006: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, Amsterdam, Netherlands, December 2006.
5. Kurniawan D, Abramson D. A WSRF-compliant debugger for grid applications. *IPDPS 2007: Proceedings of the 21st International Parallel and Distributed Processing Symposium*, Long Beach, CA, U.S.A., March 2007.
6. Kurniawan D, Abramson D. An integrated grid development environment in eclipse. *e-Science 2007: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, Bangalore, India, December 2007.
7. Kurniawan D, Abramson D. An IDE framework for grid application development. *Grid 2008: Proceedings of the Ninth International Workshop on Grid Computing*, Tsukuba, Japan, September 2008.
8. Balle SM, Hood RT. GGF UPDT User Development Tools Survey. *OGF Document Series*, October 2004. Available at: <http://www.ogf.org/documents/GFD.33.pdf> [11 August 2007].
9. Newhouse S, Schopf JM, Richards A, Atkinson M. Study of user priorities for e-Infrastructure for e-Research (SUPER), April 2007. Available at: http://www.nesc.ac.uk/technical_papers/UKeS-2007-01.pdf [15 August 2007].
10. Foster I, Kesselman C, Tuecke S. The open grid services architecture. *The Grid 2: Blueprint for a New Computing Infrastructure*, ch. 17, Foster I, Kesselman C (eds.). Morgan Kaufmann: Los Altos, CA, 2003.
11. Foster I, Kishimoto H, Savva A, Berry D, Djauoui A, Grimshaw A, Horn B, Maciel F, Siebenlist F, Subramaniam R, Treadwell J, Von Reich J. The Open Grid Services Architecture, Version 1.5. *OGF Document Series*, July 2006. Available at: <http://www.ogf.org/documents/GFD.80.pdf> [30 January 2008].
12. GDB. GDB: The GNU Project Debugger, 2007. Available at: <http://sourceware.org/gdb/> [27 August 2007].
13. NetBeans. Introduction to the NetBeans Platform, 2007. Available at: <http://www.netbeans.org/products/platform/> [27 August 2007].
14. Eclipse. Eclipse.org Home, 2007. Available at: <http://www.eclipse.org/> [29 August 2007].
15. Meyer B. Design by contract. *Advances in Object-oriented Software Engineering*, Mandrioli D, Meyer B (eds.). Prentice-Hall: Englewood Cliffs, NJ, 1991; 1–50.
16. IDB. Intel Debugger (IDB) Manual, 2008. Available at: http://www.intel.com/software/products/compilers/docs/linux/idb_manual_1.html [22 July 2008].
17. Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley: Reading, MA, 1994.
18. Riehle D. The event notification pattern—Integrating implicit invocation with object-orientation. *Theory and Practice of Object Systems* 1996; 2(1):43–52.
19. PBS. PBS GridWorks, 2008. Available at: <http://www.pbsgridworks.com/> [17 November 2008].
20. LSF. Platform Load Sharing Facility, 2008. Available at: <http://www.platform.com/Products/platform-lsf> [17 November 2008].
21. SGE. Sun Grid Engine, 2008. Available at: <http://www.sun.com/software/gridware/> [17 November 2008].
22. Garrett J. Ajax: A New Approach to Web Applications, 2005. Available at: <http://www.adaptivepath.com/ideas/essays/archives/000385.php> [26 May 2008].
23. W3C. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)—W3C Recommendation 27 April 2007, 2007. Available at: <http://www.w3.org/TR/soap12-part1/> [25 January 2008].
24. Winer D. XML-RPC Specification, 1999. Available at: <http://www.xmlrpc.com/spec/> [25 January 2008].
25. Barker R. *CASE Method: Entity Relationship Modelling*. Addison-Wesley: Reading, MA, 1990.
26. Wasserman A. Tool integration in software engineering environments. *Proceedings of the International Workshop on Environments (Software Engineering Environments)*, Chinon, France, September 1989.
27. Marlow C, Naaman M, Boyd D, Davis M. HT06, Tagging Paper, Taxonomy, Flickr, Academic Article, To Read. *Hypertext 2006: Proceedings of the Seventeenth ACM Conference on Hypertext and Hypermedia*, Odense, Denmark, August 2006.
28. WSRF. The WS-Resource Framework, 2008. Available at: <http://www.globus.org/wsrf/> [30 January 2008].
29. OASIS. OASIS Web Services Resource Framework (WSRF), 2008. Available at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf [30 January 2008].
30. Foster I. Globus Toolkit Version 4: Software for Service-Oriented Systems. *NPC 2005: Proceedings of the IFIP International Conference on Network and Parallel Computing*, Beijing, China, December 2005.
31. Foster I. A Globus Primer, August 2005. Available at: http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf [2 February 2008].
32. Sotomayor B, Childers L. *Globus Toolkit 4: Programming Java Services*. Morgan Kaufmann: Los Altos, CA, 2005.
33. HPDF. HPD Version 1 Standard: Command Interface for Parallel Debuggers, 1998. Available at: http://web.archive.org/web/20010702151435rn_1/www.ptools.org/hpdf/draft/ [19 March 2008].
34. Francioni JM, Pancake CM. A debugging standard for high-performance computing. *Scientific Programming* 2000; 8(2):95–108.
35. Ruby. Ruby Programming Language, 2008. Available at: <http://www.ruby-lang.org/> [26 May 2008].
36. Rails. Ruby on Rails, 2008. Available at: <http://www.rubyonrails.org/> [26 May 2008].
37. Rustad A. Ruby on Rails and J2EE: Is There Room for Both? July 2005. Available at: <http://www-128.ibm.com/developerworks/linux/library/wa-rubyonrails/?ca=dgr-lnxw16RubyAndJ2EE> [26 May 2008].
38. Tate B. *From Java to Ruby: Things Every Manager Should Know*. Pragmatic Bookshelf, 2006.
39. Fernandez O. *The Rails Way*. Addison-Wesley: Boston, MA, U.S.A., 2007.

40. Dierks T, Rescorla E. RFC 4346—The Transport Layer Security (TLS) Protocol Version 1.1, 2006. Available at: <http://tools.ietf.org/html/rfc4346> [3 June 2008].
41. Groovy. Groovy Home, 2008. Available at: <http://groovy.codehaus.org/> [13 June 2008].
42. Clayberg E, Rubel D. *Eclipse: Building Commercial-Quality Plug-ins* (2nd edn). Addison-Wesley: Reading, MA, 2006.
43. Gamma E, Beck K. *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*. Addison-Wesley: Reading, MA, 2003.
44. Szurszewski J. We Have Lift-off: The Launching Framework in Eclipse, 2003. Available at: <http://www.eclipse.org/articles/Article-Launch-Framework/launch.html> [16 June 2008].
45. Lee C, Matsuoka S, Talia D, Sussmann A, Mueller M, Allen G, Saltz J. A grid programming primer, August 2001. Available at: <http://www.cct.lsu.edu/~gallen/Reports/GridProgrammingPrimer.pdf> [7 February 2008].
46. Fox G, Gannon D, Thomas M. Overview of Grid Computing Environments. *Grid Computing: Making the Global Infrastructure a Reality*, ch. 20, Berman F, Fox G, Hey T (eds.). Wiley: New York, 2003.
47. Lee C, Talia D. Grid programming models: Current tools, issues, and directions. *Grid Computing: Making the Global Infrastructure a Reality*, ch. 21, Berman F, Fox G, Hey T (eds.). Wiley: New York, 2003.
48. Kielmann T, Merzky A, Bal H, Baude F, Caromel D, Huet F. Grid application programming environments. *Future Generation Grids*, Getov V, Laforenza D, Reinefeld A (eds.). Springer: Berlin, 2005; 283–306.
49. Goscinski W. IDEA: An infrastructure for the deployment of e-science applications. *PhD Thesis*, Monash University, 2006.
50. Hood R, Jost G. A debugger for computational grid applications. *HCW 2000: Proceedings of the Ninth Heterogeneous Computing Workshop*, Cancun, Mexico, May 2000.
51. Lovas R, Sunderam V. Debugging of metacomputing applications. *IPDPS 2002: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, Fort Lauderdale, FL, U.S.A., April 2002.
52. Neophytou P, Neophytou N, Evripidou P. Net-dbx-g: A web-based debugger of MPI programs over grid environments. *CCGrid 2004: Proceedings of the Fourth IEEE International Symposium on Cluster Computing and the Grid*, Chicago, IL, U.S.A., April 2004.
53. Gombas G, Marosi CA, Balaton Z. Grid application monitoring and debugging using the mercury monitoring system. *EGC 2005: Proceedings of the European Grid Conference*, Amsterdam, Netherlands, 2005.
54. Mehmood R, Crowcroft J, Hand S, Smith S. Grid-level computing needs pervasive debugging. *Grid 2005: Proceedings of the 6th International Workshop on Grid Computing*, Seattle, WA, U.S.A., November 2005.
55. Kacsuk P, Dozsa G, Kovacs J, Lovas R, Podhorszki N, Balaton Z, Gombas G. P-GRADE: A grid programming environment. *Journal of Grid Computing* 2003; 1(2).
56. Geist AL, Beguelin A, Dongarra J, Jiang W, Manchek R, Sunderam VS. *PVM: Parallel Virtual Machine*. MIT Press: Cambridge, MA, 1994.
57. MPI Forum. MPI-2 Extensions to the Message-Passing Interface, July 1997. Available at: <http://www.mpi-forum.org/docs/mpi-20.ps> [4 February 2008].
58. Gropp W, Huss-Lederman S, Lumsdaine A, Lusk E, Nitzberg B, Saphir W, Snir M. *MPI: The Complete Reference; Volume 2: The MPI-2 Extensions*. MIT Press: Cambridge, MA, 1998.
59. Allen G, Davis K, Goodale T, Hutani A, Kaiser H, Kielmann T, Merzky A, van Nieuwpoort R, Reinefeld A, Schintke F, Schutt T, Seidel Ed, Ullmer B. The grid application toolkit: Toward generic and easy application programming interfaces for the grid. *Proceedings of the IEEE* 2005; 93(3):534–550.
60. Stumpert M. g-Eclipse Architecture, 2007. Available at: <http://www.eclipse.org/geclipse/resources/D1.5.pdf> [11 January 2008].
61. g-Eclipse. g-Eclipse—Access the Grid, 2009. Available at: <http://www.geclipse.org/> [11 March 2009].
62. gLite. gLite—Lightweight Middleware for Grid Computing, 2008. Available at: <http://www.glite.org/> [2 February 2008].
63. GRIA. Service Oriented Collaborations for Industry and Commerce—GRIA, 2009. Available at: <http://www.gria.org/> [11 March 2009].
64. AWS. Amazon Web Services, 2009. Available at: <http://aws.amazon.com/> [11 March 2009].
65. Friese T, Smith M, Freisleben B. Grid development tools for eclipse. *eTX 2006: Proceedings of the Eclipse Technology Exchange Workshop in conjunction with ECOOP 2006*, Nantes, France, July 2006.
66. Friese T, Smith M, Freisleben B. GDT: A toolkit for grid service development. *GSEM 2006: Proceedings of the Third International Conference on Grid Service Engineering and Management*, Erfurt, Thuringia, Germany, September 2006.
67. Kleppe A, Warmer J, Bast W. *MDA Explained: the Model Driven Architecture: Practice and Promise*. Addison-Wesley: Reading, MA, 2003.
68. Frankel D. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley: New York, 2003.
69. Brown A. An introduction to Model Driven Architecture. Part I: MDA and Today's Systems, 2004. Available at: <http://www.ibm.com/developerworks/rational/library/3100.html> [10 January 2008].
70. Hastings S, Oster S, Langella S, Ervin D, Kurc T, Saltz J. Introduce: An open source toolkit for rapid development of strongly typed grid services. *Journal of Grid Computing* 2007; 5(4):407–427.
71. GROMACS. GROMACS: Fast, Free, and Flexible MD, 2009. Available at: <http://www.gromacs.org/> [12 February 2009].
72. Kurniawan D. ISENGARD: An infrastructure for supporting e-science and grid application development. *PhD Thesis*, Monash University, 2009.