

An IDE Framework for Grid Application Development

Donny Kurniawan and David Abramson

Monash e-Science and Grid Engineering Lab

Faculty of Information Technology, Monash University

{donny.kurniawan, david.abramson}@infotech.monash.edu.au

Abstract

Grid computing enables the aggregation of a large number of computational resources for solving complex scientific and engineering problems. However, writing, deploying, and testing grid applications over highly heterogeneous and distributed infrastructure are complex and error prone. A number of grid integrated development environments (IDEs) have been proposed and implemented to simplify grid application development. This paper presents an extension to our previous work on a grid IDE in the form of a software framework with a well-defined API and an event mechanism. It provides novel tools to automate routine grid programming tasks and allow programmable actions to be invoked based on certain events. Its system model regards resources as first-class objects in the IDE and allows tight integration between the execution platforms and the code development process. We discuss how the framework improves the process of grid application development.

1. Introduction

A grid infrastructure 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 non-trivial qualities of service [1]. 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 address 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 supplying a set of standardized interfaces to a variety of services [2]. However, to date, grid middleware gives little attention to software development issues such as compilation, deployment, and debugging.

Grid application development is concerned with the process of writing, testing, and debugging grid-based software. It is different from the traditional counterpart due to the scale and heterogeneity of the target platforms. Rather than targeting one type of computer architecture and operating system, grid software development must take into account the heterogeneous and distributed nature of grid infrastructure. Traditional programming tools are not sufficient to handle the complexity and scale of the infrastructure. Grid application development necessitates programming tools and services that are designed and engineered for grid computing. In this paper, grid applications are defined as traditional executables that can be submitted to a job scheduler, as opposed to “grid services” that must be executed in a web service container. We believe the development of traditional grid applications will continue to grow in spite of the push towards service-oriented architecture.

Recently, there is a surge of interest in grid integrated development environments (IDEs) with initiatives such as g-Eclipse [3] and GDT [4], to name a few. These projects augment IDEs such as Eclipse [5] and NetBeans [6] with plug-ins that enable the tools to handle grid application development. An integrated development environment (IDE) combines various programming tools such as a source code editor, a compiler, and a debugger into one cohesive environment [17]. The 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. We believe grid developers can gain significant benefit by using IDEs.

We have designed and implemented an infrastructure for supporting e-Science and grid application development. The infrastructure consists of a grid debugging service (GridDebug) [7], a multi-tiered programming middleware (Worqbench) [8], and an integrated grid development environment (RDT) [9]. RDT, which stands for Remote Development Tools, is a set of Eclipse IDE plug-ins that allows the IDE to be used for grid application development. This research paper presents and discusses our work on an extension to RDT. *We have augmented RDT with an IDE framework with scripting functionality.* It enables

developers to perform and automate various programming tasks, which may become cumbersome and complicated for a large number of heterogeneous grid resources. To achieve this automation, there are certain requirements that need to be met:

- The ability of the system to comprehend various entities such as resources and source file and infer the relationship between these objects.
- A runtime programmable facility that allows the developers to call or change the functionality of the IDE during its execution.

RDT has been designed and implemented to meet these requirements, specifically, the features are:

- A system model with an abstraction of grid resources and sets as first-class objects similar to files, folders, and projects.
- A system-wide tagging system for attaching metadata information to various entities such as resources and projects.
- An application programming interface (API) that enables RDT and the IDE to be accessed programmatically during runtime.
- An event notification system with user-defined callback methods.

This paper is organized as follows: Section 2 presents related work in the area of grid integrated development environment. Section 3 describes the architecture and design of the framework. The implementation details are discussed in Section 4. The discussion on how our framework improves grid software development is given in Section 5. Finally, Section 6 gives the conclusion of this research paper.

2. Related Work

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 [10], g-Eclipse [3], GDT [4], and Introduce [11].

P-GRADE (Parallel Grid Runtime and Application Development Environment) [10] is 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). 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.

g-Eclipse [3] is an integrated workbench tool for grid computing built on top of the Eclipse IDE. It has been developed by the g-Eclipse consortium which is a collaboration of several European universities and companies. g-Eclipse distinguishes three distinct user roles, namely application users, grid operators, and developers. It provides different Eclipse perspectives for each role. The user perspective allows a grid user to submit and monitor computational jobs. The operator perspective offers tools for managing and maintaining various entities such as grid resources, storage elements, and virtual organizations. The developer perspective allows a grid programmer to code, execute, and debug grid applications.

Grid Development Tools (GDT) [4] is a set of Eclipse plug-ins for service-oriented grid application development. 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 by dividing the development process into three separate model layers and automatically transforming models from one layer into the other. GDT is geared towards service-oriented grids and it supports the development of Globus Toolkit 4 and MAGE services.

Introduce [11] 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 existing tools benefit grid programmers by providing development environments that ease and simplify the process of grid application development. They take advantage of extensible non-grid IDEs to build software tools for grid computing. They implement specific grid functions such as job submission tools and resource browsers that let programmers inspect available grid resources directly from the IDE. These grid development tools provide extension points and mechanism that allow them to be extended through IDE plug-ins or enhancements. The plug-ins need to be written, compiled, and packaged according to the formal procedure as defined by the IDE.

Different to all the IDE tools described, we have designed and implemented a user-level IDE framework for grid application development. RDT has been extended with a framework with a well-defined API and an event mechanism. The framework enables RDT to be accessed programmatically and allows scriptable actions to be invoked based on certain events. These actions provide automation functions for routine grid

programming tasks. The RDT system model provides an abstraction of grid resources and sets as first-class objects in the IDE. RDT presents an alternative mechanism for improving and altering the behavior and functionality of the IDE by utilizing small and well-written scripts. There are certain advantages of using a high-level scripting language such as late binding, reflection support, expressive syntax, and dynamic typing. This scripting mechanism is different from the formal approach of extending the IDE by writing and packaging elaborate plug-ins.

3. Architecture and Design

RDT is designed on top of Worqbench [8], an integrated and modular framework for grid application development. Worqbench provides services and APIs that can be used by clients 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 accessible from the programmer's preferred development environment. Worqbench links an IDE to grid middleware in a three-tier model as depicted in Figure 1. A detailed description of Worqbench and its internal model can be found in [8][9].

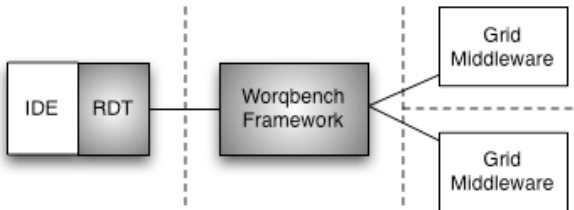


Figure 1. Worqbench three-tier model

RDT is not a replacement for an integrated development environment. Rather, it is designed as an extension to an IDE. Thus, it depends on an IDE for functions such as source code editing, project management, debugging, and so forth. It has a high-level architecture and the design is IDE-independent. RDT consists of four main components in 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 [13]. The UI tools can be customized according to the base IDE whilst retaining the underlying RDT models. The UI layer consists of two components, View and Editor, that aim to cover principal development tasks such as source code editing and resource browsing. The non-UI layer is composed of a framework model component (Core) and an application execution manager (Launch). A non-grid IDE is designed in general to handle program execution on one platform, namely, the local machine where the IDE runs. The

RDT Launch component extends the execution mechanism to handle a number of distinct remote platforms simultaneously. The architecture of RDT is depicted in Figure 2 while descriptions of the components are given as follows:

Core. The Core component specifies a framework and API with models, methods, and data structures using an object-oriented paradigm. Classes are defined to represent entities such as resources, resource sets, projects, sessions, and tasks. It is the main component of RDT which also handles the event mechanism and various event callbacks.

Launch. The Launch component is responsible for managing the execution and debugging of applications. It depends on the underlying IDE launching framework and debug model.

View. The View component is responsible for various user-interface in RDT. It provides the implementation of views such as data tables to show resources, resource sets, and projects. The component also implements preferences user-interface for RDT configuration.

Editor. The Editor component provides the implementation of a multi-view code editor. It can display multiple views of the source code based on user-given keywords or tags. The editor simplifies the process of writing grid applications for multiple architectures and operating systems. More information about the editor can be found in [9].

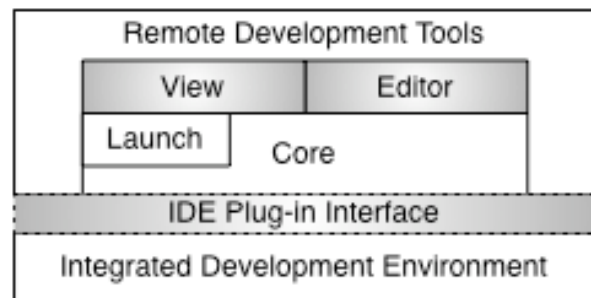


Figure 2. RDT architecture overview

Section 3.1 gives the description of the RDT framework, system model, and API. Sections 3.2 and 3.3 describe the RDT tagging system and event mechanism.

3.1 System Model and API

Traditional IDEs are geared towards local application development and normally target the platform in which they are installed on. Thus, IDEs are mainly concerned with software projects, folders, and files as opposed to target resources. RDT extends this conventional model to include grid resources and resource sets as first-class objects in the IDE. It enables

the IDE framework to recognize the relationship that exists between software projects and various grid resources; and invoke user scripts based on this association. Figure 3 depicts an abstract model and the relationship of various RDT objects.

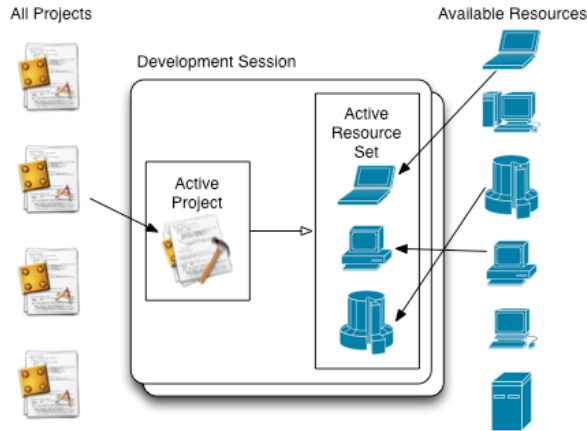


Figure 3. Abstract model of RDT

We define a development session that explicitly links an active grid resource set with an active project. An active resource set is a collection of resources where a programmer intends to develop for. The resources are selected from a pool of available machines. A project that the programmer is currently working on is called the active project. RDT operates with development sessions. With the information of the intended grid execution platforms in a session, RDT can perform various automatic or scripted programming tasks on behalf of the programmer.

A tight integration between the process of writing code and the execution platforms facilitates several development scenarios. For example, programmers can conduct performance testing on multiple platforms in a test-driven software development process and rapid prototyping of software for heterogeneous grid resources in an agile development method [16].

The framework maintains a system model with classes to represent entities such as resources, resource sets, projects, sessions, and tasks. These entities are described as follows:

Resource. It represents a distinct machine/resource in the system on where a software development project can be built and the resultant application can be executed. It is assumed that the remote machine has the necessary environment, including compilers and libraries, to build the application.

Resource Set. A number of resources can be grouped together to form a resource set or a transient grid testbed which becomes a deployment or execution target of a project. The framework defines a special set

called “All Resources” that contains all resources in the system.

Project. It represents an RDT project which is in essence an IDE-specific project with additional RDT metadata information such as user-specified tags or keywords.

Session. It represents a development session where a user writes an application in one project to be tested and executed on one resource set. RDT operations such as remote compilation, execution, and debugging require a session object.

Task. It represents a remote operation invoked on a session object. It stores the status, type, and output of the operation. For example, a user invokes a remote compilation on a session object. This operation is represented by a task object which can be inspected by the user for its attributes (e.g. status and output).

Various managers, namely ResourceManager, SetManager, ProjectManager, SessionManager, and TaskManager, handle the creation, deletion, and administration of these entities.

RDT defines a set of API methods that allows programmers to write scripts that utilize these entities and perform various IDE functions. 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 enables the programmers to conduct 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, because we use scripting, the user is able to tailor these functions to cater for variations in their particular grid infrastructure and they are not locked into a standard implementation. Importantly, the RDT framework and the API definitions are IDE-independent, however, the implementation and its language binding must follow the specification of the base IDE. A list of the currently implemented interfaces for the Eclipse IDE is given in Section 4. The implementation utilizes the Groovy programming language [12] as its language binding.

3.2 Metadata and Tagging

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 [14]. An entity can be tagged with multiple keywords which are defined arbitrarily and informally by users. 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”. Source code can also be tagged, for example, a source file for accessing a WS-GRAM service can be tagged with “gt4 webservice”. This system-wide tagging system facilitates the sharing of metadata information between source code files and grid resources; and enables the framework to understand the relationship semantics.

3.3. RDT Event Mechanism

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 mechanism based on the framework system model with user-definable callback methods. A user can write the methods to perform various checks and operations when specific events occur. For example, 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 a “test bed” set, and so forth. The event mechanism has been designed to be generic and IDE neutral. Table 1 lists the currently supported events. Several examples of the event callback methods are given and discussed further in Section 4.

ProjectChangedEvent	SetDestroyedEvent
ProjectCreatedEvent	FileChangedEvent
ProjectDestroyedEvent	FileCreatedEvent
ResourceAddedEvent	FileDestroyedEvent
ResourceRemovedEvent	FolderChangedEvent
ResourceCreatedEvent	FolderCreatedEvent
ResourceDestroyedEvent	FolderDestroyedEvent
SetCreatedEvent	

Table 1. RDT Events

4. Implementation Details

We have developed Eclipse RDT, a set of plug-ins that provide the implementation of RDT for Eclipse [5] which is an open-source IDE written in Java. Eclipse, like a number of other IDEs such as NetBeans and Visual Studio, can be augmented with new features due to its extensible plug-in system. It runs on multiple platforms where the Java Virtual Machine is available. It supports various programming languages such as Fortran, Python, Perl, and C/C++. It uses the Standard Widget Toolkit (SWT) that utilizes the native GUI libraries of the underlying operating systems. Eclipse

has the traditional IDE user-interface with multi-window editors, refactoring, syntax highlighting, and other features commonly found in advanced IDEs. Eclipse RDT is implemented according to the Eclipse plug-in guidelines [15]. It is divided into two software layers, UI and non-UI; and it consists of several Java packages.

Eclipse RDT features an embedded interpreter for the Groovy programming language [12]. Developers utilize the 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 seamlessly integrate with all existing Java objects and libraries.

The graphical user interface aspect and the multi-view code editor of Eclipse RDT have been the subject of another research paper [9] and they are not repeated here. The following two sections present the implementation of Eclipse RDT framework. Section 4.1 describes the Eclipse RDT API whilst Section 4.2 discusses the implementation of Eclipse RDT event mechanism.

4.1 Eclipse RDT API

The Eclipse RDT application programming interface is implemented as Java classes 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 Eclipse 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).
- 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.

Table 2 lists the currently implemented interfaces for the RDT entities and managers. In addition to RDT API, built-in Eclipse APIs including the SWT GUI widgets are also callable within the Groovy scripts.

```

IResource:
String getTags()
void setTags(String tags)
String getResourceName()
Integer getResourceId()
String getResourceType()
String getResourceAddr()
Integer getResourcePort()
void setResourceName(String newName)
void setResource(Integer newResource)

ISet:
String getTags()
void setTags(String tags)
String getSetName()
Integer getSetId()
IResource[] getResources()
void addResource(Integer resId)
void removeResource(Integer resId)
void refresh()
void setSetName(String newName)

IProject:
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()
IProject getProject()
ISet getSet()
void setSessionName(String newName)
void setProject(Integer newProject)
void setSet(Integer newSet)

ITask:
String getTaskName()
Integer getTaskId()
String getTaskType()
String getTaskStatus()
String getTaskOutput()
void refresh()
ISession getSession()
IResource getResource()

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 2. Eclipse RDT API

The creation, deletion, and administration of RDT entities are handled by a number of RDT managers, which are implemented as singleton objects. The managers construct objects such as `IResource`, `ISet`, `IProject`, `ISession`, and `ITask` objects, which can be used directly by calling their respective methods in an object-oriented manner. RDT entities share a number of matching attributes, specifically, name, id, and tags (with the exception of the tags attribute for the `ITask` entities). These attributes indicate the name, unique global id, and tags of the entities. It is possible to extend the API by implementing more entity attributes or by providing composite methods that perform certain higher-level functions. Eclipse RDT depends on Worqbench services for certain functions such as grid resource authentication and user authorization. The following code listing shows an example of utilizing Eclipse RDT API.

```

allResources =
    ResourceManager.getResources()
allMonashResources = [ ]

for (resource in allResources) {
    if (resource.getResourceAddr() ==~
        /monash.edu.au/)
        allMonashResources.add(resource.ge
tResourceName())
}

```

```

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: " + resName)
10
11     vSet = SetManager.getSet(setName)
12     vResource = ResourceManager.getResource(resName)
13     setTags = vSet.getTags()
14     resTags = vResource.getTags()
15
16     if (resTags.indexOf("windows") != -1 && setTags.indexOf("GT4") != -1)
17         println("Warning: adding a Windows PC to a Globus Toolkit Testbed")
18 }
19
20 def onSetDestroyed() {
21     println("Set: " + name + " has been destroyed")
22
23     allProjects = ProjectManager.getProjects()
24
25     for (project in allProjects)
26         if(project.getTags().indexOf(name) != -1)
27             cmdChangeProjectText(project.getProjectName(), "Inspect Me!")
28 }

```

Figure 4. Examples of event callback methods

```

allMonashResources.eachWithIndex() {obj,
i -> println "${i}: ${obj}"}

```

This example script prints the name of all grid resources located at Monash University. The script also demonstrates dynamic data typing, regular expressions, and closures in Groovy.

4.2 Eclipse RDT Event Mechanism

Eclipse 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. Eclipse 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 exist 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 Figure 4 with the following description:

- The method `onFileChanged()` (lines 1-6) is called when a file has been changed by the user. 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. Two special variables, `resName` and `setName`, are available within this method. On line 11, we initialize a new variable `vSet` that holds the actual Set object and on line 13, we store `vSet`'s tags in the variable `setTags`. We do the corresponding operations with the resource on lines 12 and 14. On line 16, we perform tests on the tags and print a warning if the conditional is true.
- The method `onSetDestroyed()` (lines 20-28) is called when a set is destroyed with a special variable `name` 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 text which is visible in Eclipse's Project Browser.

In the current implementation, built-in Eclipse APIs, data structures, and SWT GUI widgets are callable within the Groovy scripts. Thus, we 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.

5. Improving Grid Software Development

Different to other grid IDEs mentioned in Section 2, RDT implements a user-level IDE framework for grid application development. 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. RDT improves the process of grid software development and addresses a number of issues posed by the process such as scalability, efficiency, quality control, and extensibility.

There are many aspects of scalability such as performance, fault tolerance, load management, and code maintainability. However, there is one aspect of scalability that is often overlooked, namely, the ability of the development process to scale as the number of grid resources increases. The RDT framework and its grid system model enable developers to automate many tedious grid programming tasks for a large number of resources. Another aspect related to scalability is efficiency. RDT streamlines grid application development by providing a mechanism that allows programmable actions to be invoked based on certain conditions or grid resources. In addition, the RDT framework and its event mechanism enable check and verification steps to be performed on source code in relation to grid resources and their attributes or tags. The use of a high-level scripting language to drive the development tools is also advantageous since the user can customize the tool functions to cater for variations in their particular grid infrastructure.

We believe the framework improves and advances the state of grid programming tools. It simplifies and eases the process of developing grid software for a large number of heterogeneous and distributed resources.

6. Conclusion

Grid application development requires sophisticated programming tools that can handle the complexity inherent in grid infrastructure. Programming tools such as grid development environments have been proposed and implemented. These tools can be further improved by incorporating scripting functionality that allows programmers to perform complex development tasks for a large number of heterogeneous grid resources. This research paper

presents an extension to our earlier work, RDT, in the form of an IDE framework. The improvement permits the grid IDE to be scriptable to facilitate the automation of routine and tedious grid programming tasks. The grid system model integrates the execution platforms with the software development process. The paper also presents Eclipse RDT, which is an implementation of the framework for the Eclipse IDE.

RDT can be further improved by defining more event types and API methods. It is possible to improve and simplify the current Eclipse RDT implementation by providing buttons for toolbars and menus that allow users to call and invoke pre-packaged or frequently used RDT scripts directly.

7. References

- [1] 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.
- [2] 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. Wiley and Sons, 2003.
- [3] g-Eclipse - Access the Grid, <http://www.geclipse.org>.
- [4] Thomas Frieze, Matthew Smith, and Bernd Freisleben. GDT: A Toolkit for Grid Service Development, *International Conference on Grid Service Engineering and Management*, 2006.
- [5] Eclipse, <http://www.eclipse.org/>.
- [6] NetBeans, <http://www.netbeans.org/>.
- [7] Donny Kurniawan and David Abramson. A WSRF-Compliant Debugger for Grid Applications. *Parallel and Distributed Processing Symposium*, 2007. IPDPS 2007. IEEE International. March 2007.
- [8] 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*, December 2006.
- [9] Donny Kurniawan and David Abramson. An Integrated Grid Development Environment in Eclipse. In *e-Science 2007: Proceedings of the Third IEEE Conference on e-Science and Grid Computing*, December 2007.
- [10] 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*, Volume 1, Issue 2, 2003, Pages 171 - 197.
- [11] 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*, Volume 5, Issue 4, 2007.
- [12] Groovy. <http://groovy.codehaus.org/>.
- [13] Anthony I. Wasserman. Tool integration in software engineering environments. *Lecture Notes in Computer Science - Software Engineering Environments*, Volume 467, Pages 137-149, 1990.
- [14] Cameron Marlow, Mor Naaman, Danah Boyd, and Marc Davis. HT06, tagging paper, taxonomy, Flickr, academic article, to read. In *Proceedings of the Seventeenth Conference on Hypertext and Hypermedia*, 2006.
- [15] Eric Clayberg and Dan Rubel. *Eclipse: Building Commercial-Quality Plug-ins*, second edition, Addison Wesley, 2006.
- [16] Craig Larman and Victor Basili. *Iterative and Incremental Development: A Brief History*. *IEEE Computer*, Volume 36, Issue 6, 2003.
- [17] F. A. Case. Computer-aided software engineering (CASE): technology for improving software development productivity. *ACM SIGMIS Database*, Volume 17, Issue 1, Fall 1985.