

An Integrated Grid Development Environment in Eclipse

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

With the proliferation of Grid computing, a large number of computational resources are available for solving complex scientific and engineering problems. Nevertheless, it is non-trivial to write, deploy, and test Grid applications over heterogeneous and distributed resources. Further complicating matters, programmers may need to manually manage variations in source code due to resource heterogeneity. This paper presents an implementation of an integrated Grid development environment that leverages IBM's Eclipse IDE and our application development framework, Worqbench. It provides novel tools to develop and debug Grid software. It regards resources as first-class objects in the IDE and allows tight integration between the test beds and the code development process. We discuss how the environment assists programmers in developing Grid applications.

1. Introduction

Grid computing allows widely distributed resources such as scientific instruments, clusters, and databases to be tied together to give the appearance of a single large system [1]. It enables e-Science and future research on large-scale problems, which require high-performance computing and terabytes of data storage. This requirement may only be addressed collectively by several organizations. Grid computing requires particular software, Grid applications to take advantage of the underlying architecture and resources. A Grid test bed may consist of heterogeneous resources with different architectures and operating systems and this variability must be taken into account by the applications. Apart from heterogeneity, additional considerations also include issues such as security, resource management, information services, and data management [2]. To mitigate some of these concerns, Grid middleware has been developed. This middleware provides a set of common services for information, security, and management. However, to date, Grid middleware gives little attention to software development issues such as compilation, deployment, and debugging.

Due to the distributed and heterogeneous nature of test beds, developing Grid applications is more complex than for traditional local applications. Grid programming is complicated by the lack of sophisticated tools as described in a quote by Balle and Hood in 2004 [3], “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.”

There is a recent surge of interest that addresses the issue of Grid development tools, with initiatives such as g-Eclipse [4] and GDT [5]. These projects utilize extensible IDEs (e.g. Eclipse [6] and NetBeans [7]) to provide powerful and cohesive development environments for the benefit of both Grid programmers and users. IDEs have a steep learning curve and are difficult to master [8][9]. However, various studies have shown that there is an increase in programmer's productivity once a particular environment has been mastered [10][11][12]. IDEs improve productivity in three areas. First, in the area of quality, they provide powerful tools, for example, an automatic syntax checker. Second, through features such as automatic recompilation and source code tracking in the debugging phase, IDEs improve productivity in the area of code management. Third, they provide functions such as code-generation support and refactoring that can enhance programmer's efficiency [10][13].

We believe Grid developers can gain tremendous benefit by using IDEs. On the other hand, current “Grid-enabled” IDEs are not as feature rich as their traditional counterparts. For example, debugging Grid applications is still not supported. 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 (SOA).

This research paper presents an integrated Grid development environment implemented on top of IBM's Eclipse IDE [6] and our application

development framework, Worqbench [14]. It differs from other Grid IDEs by treating Grid resources as first-class objects in the environment and by providing tight integration between the software development process and the test beds. It supports debugging Grid applications by utilizing our Grid debugging service [14][15]. It also implements a multi-view source editor that simplifies programmers' task in writing code for multiple heterogeneous platforms.

The rest of the paper is organized as follows. Section 2 presents Worqbench and Eclipse as the underlying technologies. The architecture and implementation details are described in sections 3 and 4 respectively. In section 5, we outline related work in the area of Grid development environment. Section 6 presents the conclusion.

2. Worqbench and Eclipse

Grid middleware has defined common protocols and services in the domains of information, security, and management. However, common services that support application development do not currently exist. To this end, we have developed a new system called Worqbench [14] which complements existing middleware by providing services for Grid software development.

Worqbench is an integrated and modular framework for application development on the Grid. It is IDE and middleware independent. Thus, Worqbench allows any IDE with appropriate plug-ins to be used with an existing Grid middleware. It provides services and APIs that can be used by clients to package, launch, and debug Grid applications. It has a layered architecture that consists of several main components and a set of modules as interfaces to various middleware and IDEs. The framework links an IDE to Grid middleware in a three-tier model as depicted in Figure 1.

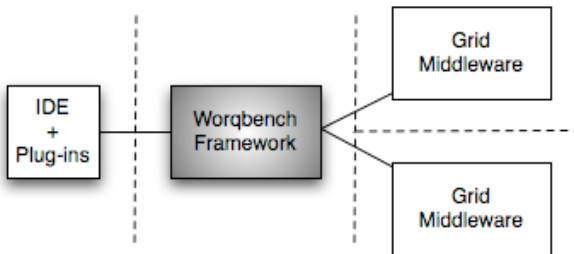


Figure 1. Three-tier model

Worqbench allows the aggregation of Grid resources with different middleware and architectures into a single Grid test bed accessible from the programmer's preferred development environment. The architecture of the framework is shown in Figure 2.

The key services are namely packaging, launching, and editing services.

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

Launching Service: The Launching Service manages the execution of an application on Grid resources. It utilizes the underlying Grid middleware job submission tools. The service supports two modes of execution: normal and debug. The debug mode utilizes a middleware level Grid debugging service [15]. Output from the application is forwarded by the launching service to Worqbench clients. A generic command is provided that allows a client to control the execution of a debugged application.

Editing Service: The Editing Service provides the management of a software project at the file system level. It allows a user to retrieve and to query individual project folders and files. Metadata information of the folders and files such as entry name, parent name, and type can also be queried by using the service.

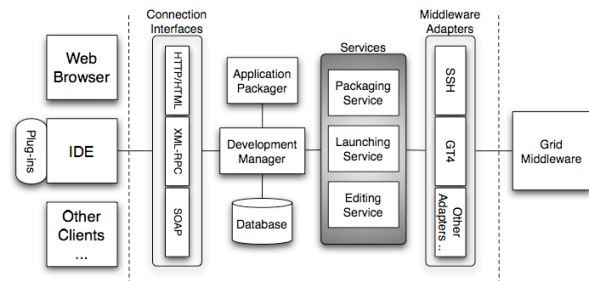


Figure 2. Key components in Worqbench

The description of other components and the internal model of the framework can be found in [14]. Worqbench also provides a web interface portal for administrative tasks such as defining new resources and test beds. However, a web browser is not a suitable environment for application development that requires heavy interaction such as source code editing and debugging which are better provided in an IDE.

We have implemented plug-ins for IBM's Eclipse IDE that utilize Worqbench services. Eclipse is an IDE written in Java. It was originally developed by IBM which gave the code to the Eclipse Foundation under an open-source license. Eclipse 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. Eclipse's GUI performance is fast compared to Java Swing applications because it uses Standard Widget Toolkit (SWT) that utilizes the native GUI libraries of the underlying operating systems. Eclipse supports various programming languages such as PHP, Fortran, Python, and Perl. It also provides an extensive C/C++ support through the CDT (C/C++ Development Tooling) plug-ins. Eclipse has the traditional IDE user-interface with multi-window editors, refactoring, syntax highlighting, and other features commonly found in advanced IDEs.

By implementing Grid development functions as Eclipse plug-ins, traditional application programmers accustomed to Eclipse can develop Grid software without learning another development environment.

3. Architecture of the development environment

Remote Development Tools (RDT) is a set of Eclipse plug-ins that leverage Worqbench services for Grid application development. It consists of four principal components, namely Core, Launch, Editor, and View. The design follows the Eclipse plug-in guidelines [16] by separating the non-UI components (Core and Launch) from the UI components (Editor and View). The architecture of the plug-ins is depicted in Figure 3 while the descriptions are as follows.

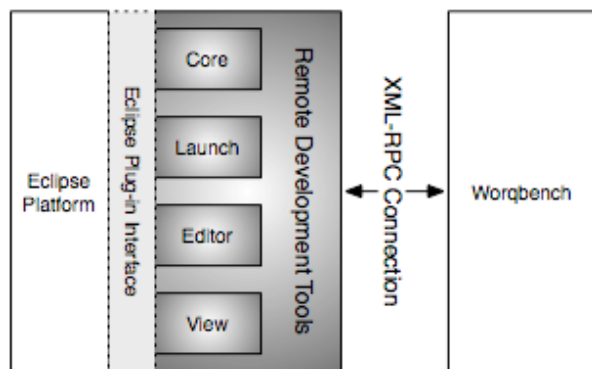


Figure 3. Architecture overview

Core: The Core component is the main link between RDT and Worqbench. All calls to Worqbench services go through the Core component. It provides controllers for Eclipse users to manage resources, resource sets, and Grid application projects.

Launch: The Launch component is responsible for the execution and debugging of a Grid application. It is linked with the built-in Eclipse launching framework and debug model. The component communicates with Worqbench's packaging and launching services.

Editor: The Editor component provides editing service for source files in RDT. The component utilizes

low-level Worqbench APIs to implement a multi-view code editor. It can display multiple views of the source code based on keywords given by a user.

View: The View component is responsible for various user-interface in RDT. It implements table views to show resources, resource sets, and projects. It provides console windows and Eclipse preference windows for RDT configuration.

3.1. Internal model

Worqbench maintains an internal model of the system and it defines entities such as *resources*, *resource sets*, *projects*, and *sessions* [14]. RDT follows this system model according to the Model-View-Controller (MVC) design pattern. RDT implements controllers in the *Core* component and the corresponding user-interface in the *View* component for the entities. Unlike Worqbench, RDT does not have a *user* entity since Eclipse is a single-user programming environment.

The *resource* entity represents one distinct machine/resource in the system. A programmer can form a Grid test bed by grouping several resources into a *resource set*. A Grid application project is represented by the *project* entity. The *session* entity represents an RDT development session where the programmer works with a *project* to be tested and executed on a *resource set*.

Traditionally, an IDE deals only with files, folders, and projects as distinct logical objects in the system. It does not distinguish resources since it can only handle one resource, namely the computer where the IDE runs. However, Worqbench and RDT allow a programmer to handle multiple resources from the IDE. By defining *resources* and *resource sets* as part of the system model, the target machines become integral in the IDE. The resources become first-class objects similar to files, folders, and projects. This integration exposes resource heterogeneity in a controlled manner in the software development process. It assists the programmer in writing Grid applications for multiple platforms.

4. Implementation and discussion

RDT is implemented in Java. It uses the XML-RPC interface to communicate with Worqbench. The Core component, which is an XML-RPC client, translates the requests and responses from XML to Java objects required by the plug-in. Worqbench URL address and the user credentials can be specified in the Eclipse preferences window. Section 4.1 describes the implementation of common IDE features for Grid application development, for example debugging and console output. Section 4.2 discusses tagging as a

mean of providing metadata information in RDT. Section 4.3 presents a multi-view code editor that aids Grid programmers in code development for heterogeneous resources.

4.1. Common IDE features

We have developed a Grid debugging service [15] that is compliant with the Web Service Resource Framework (WSRF), which is the standard for Grid middleware architecture [17][18]. The service complements existing middleware by providing a standard set of application programming interfaces (APIs) suitable for debugging and testing computational Grid applications. These debug APIs are utilized for debugging by the Worqbench Launching Service.

The RDT Grid debugging plug-in acts as an intermediate layer between Eclipse and Worqbench debug models. It translates, for example debug events and breakpoints between the two models. The plug-in is currently adapted for C and C++ applications due to Worqbench's limited support for other languages. However, the plug-in extends the generic language-independent Eclipse platform debug model rather than the Eclipse CDT debug model, thus, the difficulty in adapting it to support other languages is mitigated.

The RDT Grid debugging plug-in, which is a part of the RDT Launch component, allows a programmer to debug Grid applications running on remote resources from the Eclipse IDE on the programmer's desktop. It supports common debugging operations such as inspecting variables; adding or removing breakpoints; stepping or resuming the execution; code highlighting of the current line; and so forth. An example of a debugging session is depicted in Figure 4.

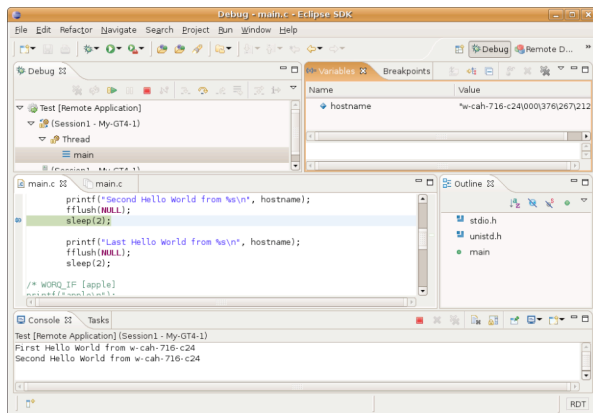


Figure 4. Grid debugging service in Eclipse

To execute a Grid application, the programmer opens the Eclipse launch configuration dialog as

depicted in Figure 5. The programmer needs to specify the development session, the Grid application project, and the resource set on which the application will be executed. There is also an option to perform remote build before the execution.

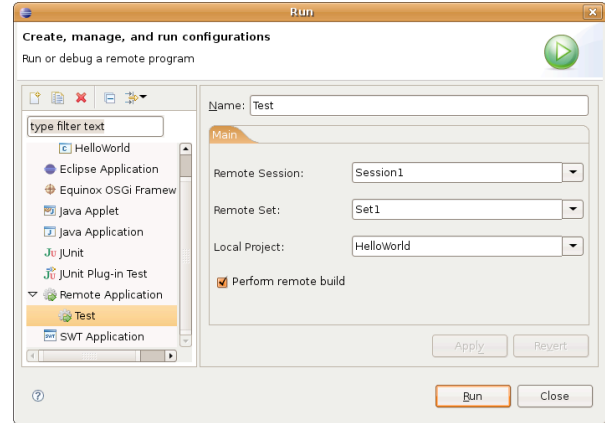


Figure 5. Eclipse launch configuration dialog

The output from executing or debugging a Grid application is displayed in the Eclipse console windows. Each window displays the program output from one resource. The console window also accepts user input for the application.

RDT has its own Eclipse perspective, the Remote Development perspective that displays windows and editors related to Grid application development. RDT implements various Eclipse views that provide management functions for remote resources, resource sets, and sessions. Figure 6 depicts the RDT perspective.

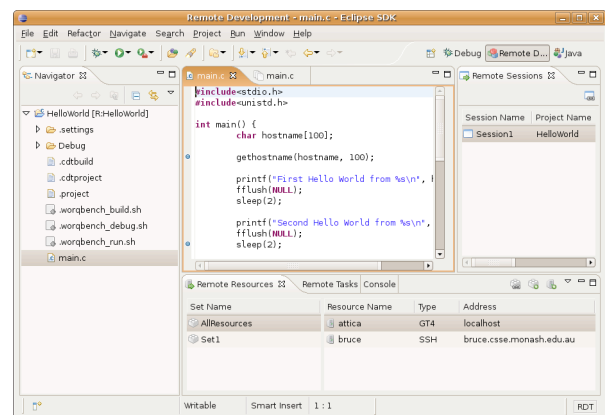


Figure 6. RDT perspective

4.2. Metadata and tagging

Although Grid middleware defines common security, management, and information services and

offers a uniform way for executing Grid applications, it does not solve the problem of architectural heterogeneity. The problem stems from the fact that Grid test beds may consist of a mix of resources with different operating systems and architectures, which must be taken into account by application developers. There are two general approaches. One approach is to employ a software layer to manage the underlying differences. This way requires the applications to be written in high-level languages that have portable virtual machines and interpreters such as Java and Python. However, it may not fully exploit advantages offered by some specific computer architectures and in practice, majority of Grid applications are still written in system languages such as C/C++ or Fortran. The second approach, which is taken by this research paper, is to expose the differences (i.e. the system properties of the resources) but provide tools for managing the heterogeneity. Developers can write applications in low-level languages and use these tools to assist them in tailoring the code to specific architectures.

Worqbench and RDT employ tagging to describe information about entities such as resources, resource sets, and development projects. 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 [19][20]. An item can be tagged with multiple keywords which are defined arbitrarily and informally by users. For example, a Sun workstation can be tagged with “sun sparc solaris” or a Linux machine can be tagged with “i686 redhat kernel-2.6.20 smp development-machine”.

Recently, tagging systems have grown in popularity on the web [20]. Many web applications such as Flickr, Delicious, and Google allow users to tag resources (e.g. photos, bookmarks, and emails) with their own keywords. Tagging is user-centric or consumer-centric as opposed to producer-centric. It lets users to associate values or properties as perceived by them to the resources. As far as we are aware of, tagging has never been applied in relation to computer machines and test beds. In the context of Grid computing, this approach has the potential to improve discovery and information systems for Grid resources. Programmers and users will be able to discover “soft facts” (e.g. ranking, reputation, popularity, etc.) in addition to “hard facts” (e.g. operating system versions, architecture types, memory sizes, etc.). We plan to develop interfaces to various information services such as Globus Toolkit Monitoring and Discovering Systems (MDS) to populate tag fields of resources with factual information automatically and without any user actions, for example: “machinetype:i686-pc ostype:linux-

gnu”. Currently, this information must be entered manually in tag fields.

The idea of tagging, however, is not new in the domain of software development. Conditional compilations and “`ifdefs`” which are the traditional ways to mark portions of code with metadata information can be considered as tagging systems. For example in “`#ifdef WIN32`”, the keyword “WIN32” can be regarded as a tag, thus, tagging is a natural and familiar concept to application developers.

A system-wide tagging for source code and resources is employed in Worqbench and RDT. It opens up possibilities for various interactions between application source code and target resources. For example: highlighting source files that need to be updated when a new resource is added to the test bed by looking at the resource tags, listing code portions that will not be executed in selected computers by examining the resource and code tags, and so forth.

4.3. Multi-view code editing

It is common for Grid programmers to write software for multiple platforms. However, it is not a trivial task. Different operating systems and computer architectures may introduce different function calls, arguments, and interfaces. Furthermore, some platforms may not have functions provided by others. Writing multi-platform applications introduces variations in the source code that are typically handled in three ways: project level, file level, and code level. At the project level, a programmer creates one distinct application project for each platform. At the file level, the variations are handled by creating different source files, e.g. `utility-win32.c` and `utility-unix.c`. Files common for one platform can then be placed in one directory/folder. The third way, which is the typical method, is to put the variations in the source code itself by employing pre-processor and macro directives, e.g. `#define` and `#ifdef`. A programmer normally employs a mix of the second and third methods.

Macro and pre-processor directives are difficult and error-prone to manage [21]. Adding a new unique platform requires the programmer to find, inspect, and possibly change every occurrence of the directives. In addition, directives also make understanding the source files difficult because of the additional code variations [21]. Various tools and editors have been developed in the software engineering domain to manage such directives [22][23][24].

In contrast to existing approaches, we have developed a multi-view code editor for managing code variations based on the concept of tagging. The editor can show and hide source code depending on user-given keywords or tags. It has one base view and

multiple sub views. The base view shows the full raw source code while the sub view shows only the relevant lines of the code based on the tag. An example to illustrate this is given below.

We define two new directives `/* WORQ_IF ... WORQ_END */` and `// WORQ_IF ... // WORQ_END` as code markers. We avoid overloading `#define` and `#ifdef` since they can also be used by programmers for non-platform-variation purposes, e.g. for a debugging purpose. `WORQ_IF` accepts a Boolean expression as the condition, for example `“!linux && (sparc || i386)”` marks the code that is relevant when the tag contains `sparc` or `i386` but not `linux`. Consider, for example, the following source file:

```
#include<stdio.h>
int main() {

/* WORQ_IF [linux && sparc]
printf("Linux on a Sparc machine\n");
WORQ_END */

// WORQ_IF [windows]
printf("Windows\n");
// WORQ_END

return 0;
}
```

When the file is opened with the tag “`linux sparc`”, the editor opens a new sub view with the following code:

```
#include<stdio.h>
int main() {

printf("Linux on a Sparc machine\n");

return 0;
}
```

If the given tag is “`linux`”, the editor shows the main body with only the return statement since the condition of the `WORQ_IF` is “`linux && sparc`”. Similarly, when the given tag is “`windows`”, the sub view shows:

```
#include<stdio.h>
int main() {

printf("Windows\n");

return 0;
}
```

The editor works with a single instance of the file and it does not keep a separate copy of the file for each sub view. A programmer can edit the source text in a view and the changes will be saved in the file. The problem of concurrency is avoided by allowing only one view to be active and editable at one time. Figure 7 depicts the base view of the editor showing the raw source code. Figure 8 depicts the sub view of the editor with the tag “`linux sparc`”.

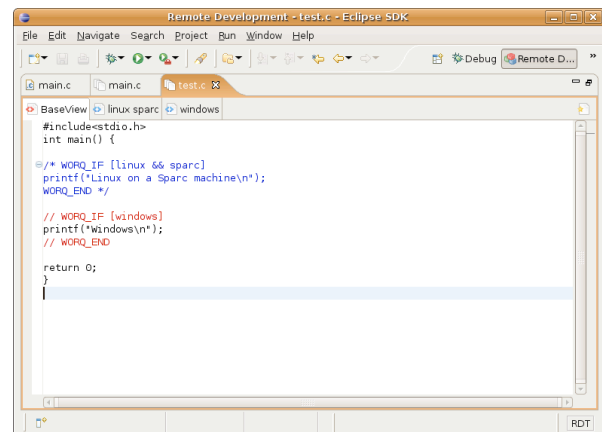


Figure 7. Base view of the editor

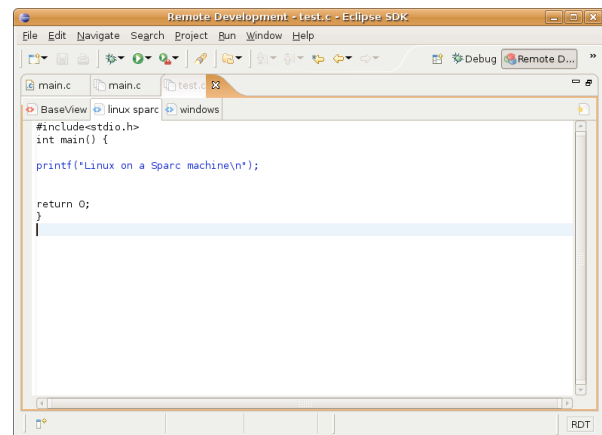


Figure 8. Sub view of the editor

5. Related work

There are several integrated development tools which are specifically designed for writing Grid applications. Some examples of them that represent the current state of research are: GT4IDE [25], GriDE [26], Grid Development Tools [5], and g-Eclipse [4].

GT4IDE [25] is an integrated development environment specifically suited to the needs of Globus Toolkit 4 (GT4) programmers. It is built on top of Eclipse IDE. The main target of GT4IDE is the development of Grid applications as services specifically GT4 services. It presents an environment

that seamlessly integrates all the steps from coding to deployment. It provides the necessary build tools and automatic generation of WSDL (Web Services Description Language) files for GT4 Grid services. Due to the early stage of the implementation, GT4IDE still lacks necessary features such as a Grid debugger and profiler.

GriDE [26] from Asia Pacific Science & Technology Center (APSTC) is a Grid-enabled IDE utilizing NetBeans as the base IDE. NetBeans [7] is an open-source IDE from Sun Microsystems. Similar to Eclipse, it has standard features commonly found in traditional IDEs: refactoring support, syntax highlighting, visual editors, and an extensible plug-in system. GriDE provides a traditional programming environment through NetBeans and a set of Grid-specific tools: a simple Grid workflow editor, a resource browser, a job submission tool, and a job monitoring tool. GriDE's Grid workflow editor can be used to construct simple workflow like batch job and single job executions. Users can browse available Grid resources using the resource browser and with the job submission tool users can submit Globus Toolkit 2 jobs directly from the IDE. Functionalities like Grid debugging and complex Grid workflows are still unimplemented.

Grid Development Tools (GDT) [5] is a set of plug-ins for service-oriented application development in the Eclipse IDE. It is part of the Marburg Ad-hoc Grid Environment (MAGE) which is a WSRF-compliant Grid middleware solution developed in the University of Marburg in Germany. GDT 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. Similar to GT4IDE, GDT is geared towards Grid services and it supports GT4 and MAGE services. GDT utilizes the standard remote Java debugging in Eclipse to support interactive service debugging.

The g-Eclipse [4] 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 provides three distinct Eclipse perspectives for Grid users, operators, and developers. The user perspective allows a Grid user to submit and monitor computational jobs. The operator perspective offers tools for managing and maintaining Grid resources, storage elements, and virtual organizations. The developer perspective allows a Grid programmer to code, execute, and debug Grid applications. g-Eclipse is still under active development, the latest prototype release contains Grid management tools for authentication, resources, jobs, and data.

Different to all the tools described, our approach consists of two parts, Worqbench and RDT.

Worqbench implements the back-end services and APIs for Grid application development while the user-interface and high-level tools are provided by RDT in the form of Eclipse plug-ins. This modular arrangement allows us to port RDT to other IDEs such as NetBeans without much difficulty. It also enables RDT to work with new Grid middleware without any modification provided that the Worqbench interface to the middleware has been implemented.

6. Conclusion

This research paper presents the design and implementation of RDT, an integrated Grid development environment. It allows e-Scientists and programmers to develop, execute, and test Grid applications on remote resources from local desktops. RDT utilizes Eclipse as the base IDE and leverages Worqbench services for Grid development functions. It provides novel tools to write and debug Grid software.

RDT addresses the challenge of writing software for heterogeneous resources by integrating the test beds with the code development process. In RDT, resources are regarded as first-class objects. It exposes the resource heterogeneity in a controlled manner. RDT provides a tagging system and a multi-view code editor to assist Grid developers in handling the heterogeneity.

RDT, however, is an ongoing research project. Interfaces to various information services such as Globus Toolkit Monitoring and Discovering Systems need to be developed for the tagging system. Even though the implementation of the multi-view code editor is complete, the build and runtime systems still need to be modified to understand the `WORQ_IF` directive.

7. References

- [1] Foster, I., Kesselman, C. and Tuecke, S., The anatomy of the grid: enabling scalable virtual organizations, International Journal of Supercomputer Applications, 2001.
- [2] Joseph, J. and Fellenstein, C., Grid Computing, Prentice Hall, 2004.
- [3] Balle, S. M. and Hood, R. T., GGF UPDT User Development Tools Survey, March 2004.
- [4] g-Eclipse - Access the Grid, <http://www.geclipse.org>.
- [5] Friese, T., Smith, M. and Freisleben, B., GDT: A Toolkit for Grid Service Development, International Conference on Grid Service Engineering and Management, 2006.
- [6] Eclipse, <http://www.eclipse.org/>.
- [7] NetBeans, <http://www.netbeans.org/>.
- [8] Reis, C. and Cartwright, R., Taming a Professional IDE for the Classroom, Proceedings of the 35th

SIGCSE Technical Symposium on Computer Science Education, 2004.

[9] Seffah, A., Rilling, J., Investigating the Relationship between Usability and Conceptual Gaps for Human-Centric CASE Tools, p. 226, IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01), 2001.

[10] Case, A. F., Computer-aided software engineering (CASE): technology for improving software development productivity, ACM SIGMIS Database, Volume 17, Issue 1, Fall 1985.

[11] Tsuda, M., Morioka, Y., Takadachi, M. and Takahashi, M., Productivity analysis of software development with an integrated CASE tool. Proceedings of the 14th international conference on Software engineering, 1992.

[12] Glass, R. L., The realities of software technology payoffs. Communications of the ACM. Volume 42 , Issue 2, February 1999.

[13] Frank, R., An Inherent Conflict in Using IDEs in Computer Language Courses. The Proceedings of ISECON 2002, v 19 (San Antonio): 221d. ISSN: 1542-7382, 2002.

[14] Kurniawan, D., Abramson, D., Worqbench: an Integrated Framework for e-Science Application Development, e-Science meeting 2006, Netherlands, December 2006.

[15] Kurniawan, D., Abramson, D., A WSRF-Compliant Debugger for Grid Applications, Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International 26-30 March 2007.

[16] Clayberg, E. and Rubel, D., Eclipse: Building Commercial-Quality Plug-ins, second edition, Addison Wesley, 2006.

[17] Foster I., Kesselman C., Nick J.M. and Tuecke S., The physiology of the Grid: an open grid services architecture for distributed systems integration, Globus Project, 2002.

[18] Czajkowski, K., Foster, I., Frey, J., et al. The WS-Resource Framework, Version 1.0. <http://www.globus.org/wsrf/>.

[19] Tag (metadata) - Wikipedia, the free encyclopedia, <http://en.wikipedia.org/wiki/Tags>.

[20] Marlow, C., Naaman, M., et al. HT06, tagging paper, taxonomy, Flickr, academic article, to read, Proceedings of the seventeenth conference on Hypertext and hypermedia, 2006.

[21] Ernst, M. D., Badros, G. J., and Notkin, D. An empirical analysis of C preprocessor use, IEEE Transactions on Software Engineering, 28(12):1146–1170, Dec. 2002.

[22] Kruskal, V., A Blast from the Past: Using P-EDIT for Multidimensional Editing, Workshop on Multi-Dimensional Separation of Concerns in Software Engineering, ICSE 2000.

[23] Livada, P.E., Small, D.T., Understanding Code Containing Preprocessor Constructs, Proc. IEEE Third Workshop Program Comprehension, pp. 89–97, Nov. 1994.

[24] Kuhlback B., Riediger, V., Folding: An Approach to Enable Program Understanding of Preprocessed Languages. In E. Burd, P. Aiken, and R. Koschke, editors, Proc. of Eighth Working Conference on Reverse Engineering; WCRE'01, pp. 3–12, IEEE Computer Society, 2001.

[25] Globus Service Build Tools - GT4IDE, <http://gsbt.sourceforge.net/>.

[26] GriDE, <https://gride.dev.java.net/>.