

Worqbench: an Integrated Framework for e-Science 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

With the proliferation of Grid computing, potentially vast computational resources are available for solving complex problems in science and engineering. However, writing, deploying, and testing e-Science applications over highly heterogeneous and distributed infrastructure are complex and error prone. Further complicating matters, programmers may need to target a variety of different Grid middleware packages. This paper presents the design and implementation of Worqbench, an integrated, modular and middleware neutral framework for e-Science application development on the Grid. Worqbench can be incorporated into a number of existing Integrated Development Environments, further leveraging the advantages of such systems. We illustrate one such implementation in the Eclipse environment.

1. Introduction

"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." [1]

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 [2]. It has much to benefit from Grid computing which couples geographically distributed resources such as processors, data stores and scientific instruments to act as a single system [3]. Utilizing Grid resources requires appropriate "Grid-enabled" software, and the lack of such software has impeded the uptake of the new technology to date. e-Science

applications need to be specifically written to take the advantage of Grid infrastructure.

One of the most significant issues is that Grid software must manage the heterogeneity that is present in the underlying fabric. Specifically, because Grid resources may have different architectures and operating systems, software needs to take account of this variability. Thus, traditional applications that target only one computing platform cannot be easily deployed and executed on all resources in a Grid system involving multiple computer architectures. In addition to managing heterogeneity, developers need to handle issues such as security, resource management, information services, and data management [4]. To relieve programmers from some of these concerns, the Grid community has developed specific middleware that provides a uniform set of services for security, information, and management. Grid middleware, however, gives little attention to architectural variability in the platforms, as well as software development issues such as compilation, deployment, debugging, and profiling.

Currently, e-Scientists and Grid developers typically rely on disintegrated CLI (command-line interface) tools [1]. Specifically, programmers write applications on a local machine and then transfer them to Grid nodes for execution. Debugging is usually performed by logging into the execution nodes, using a range of techniques from displaying program state with `print` statements through to running a command line debugger such as GDB [1][5]. Applications also need to be compiled and built according to the platform of the targeted nodes, and this again requires a user to log in to build the software. These software development techniques are time consuming, error prone and inefficient, and do not scale to large Grids.

In contrast, in the traditional non-Grid environment, developers have access to a range of powerful development tools and platforms. For example, Integrated Development Environments (IDEs) combine various programming tools into one cohesive

environment. Although IDEs have a steep learning curve and can be difficult to master [6][7], various studies have shown that once a particular environment has been mastered, it can significantly increase programmers' productivity [8][9][10]. IDEs are effective because they improve productivity in three areas, namely efficiency, management control, and quality [8][11]. First, they provide functions, such as refactoring and code-generation support, that can substantially reduce the amount of time required to develop an application. Second, they provide programmers with better direct and automated control over the development process through features such as automatic recompilation and source code tracking in the debugging phase. Third, they improve code quality by providing tools such as an automatic syntax checker.

We believe that e-Scientists and Grid developers have much to gain from using IDEs. However, almost all current platforms are geared towards traditional application development on a local machine, and thus do not address many of the peculiar aspects of Grid software development that make it difficult. To fill this gap, we aim to provide a framework and a set of IDE "plug-ins" that make IDEs suitable for Grid application development. In the context of this paper, we define Grid applications as traditional executables that can be run or submitted to a job scheduler rather than as "Grid services". In spite of the push towards service-oriented architecture, we believe the development of traditional Grid applications will grow and coexist with the development of Grid services.

The framework requires standards that are lacking in Grid software development. For example, whilst common protocols and standards have been defined in the domains of security, information, and management, no equivalent standards exist in the area of debugging or profiling. Over and above defining new software development based standards, and implementing them in middleware, we plan to build a set of tools that leverage the new services.

This research paper focuses on the design and implementation of an integrated framework of Grid development tools that bridges the gap between traditional IDEs and Grid middleware. This paper is organized as follows: section 2 presents current research in the area of Grid application development tools. In section 3, we outline the motivation and objectives of the framework, Worqbench. Section 4 describes the architecture of the framework. The implementation details of the framework are discussed in section 5. Section 6 gives the summary and the conclusion of this research paper.

2. Related Work

In the area of Grid software development, researchers have developed several programming tools and IDEs which are specifically designed for writing Grid applications. Indicative examples that represent the current state of research are: P-GRADE [12], GrADS/VGrADS [13][14], GT4IDE [15], and GriDE [16].

P-GRADE (Parallel Grid Runtime and Application Development Environment) from MTA SZTAKI in Hungary provides a high-level graphical environment to develop parallel applications for parallel systems and the Grid [12]. It consists of an editor (Gred) for a graphical language (Grapnel), a debugger (Diwide), a profiler (Prove), and application monitoring tools (GRM and Mercury). Grapnel is a hybrid programming language that uses both textual and graphical representations to describe the application. In P-GRADE, rather than directly write the application in C/C++ or Fortran, developers write the program graphically in Grapnel using Gred. P-GRADE converts the Grapnel code to C/C++ or Fortran code and compiles it. The resulting application can be submitted to resources as Condor, Condor-G, or Globus 2 jobs.

The GrADS (Grid Application Development Software) project focuses on addressing the fundamental problem of programming in the highly complex and dynamic Grid environment [13]. The GrADS software toolkit (GrADSoft) is a new approach completely different from the traditional development cycle of separate code, compile, link, and execute stages. In the GrADSoft architecture, a continuous process of adapting applications to a changing Grid replaces the normal steps of program creation, compilation, execution, and post-mortem analysis. This continuous process requires a specific execution environment, a new compiler and a new runtime system. Even though GrADS provides the necessary software development tools, existing applications need to be rewritten to fully take the advantage offered by GrADS. VGrADS (Virtual GrADS) [14] aims to provide high-level hierarchical abstract view of Grid resources for application use. The abstraction is implemented by Virtual Grids that provide a clean separation of concerns between Grid applications and the complexity of the Grid infrastructure. The application specifies its resource needs and the VGrADS execution system finds and allocates appropriate resources. GrADS/VGrADS is an ongoing research project and currently the software tools are still in the prototype stage. It still remains to be seen whether the new approach by GrADS/VGrADS can

simplify the process of software development on the Grid.

GT4IDE is an integrated development environment specifically suited to the needs of Globus Toolkit 4 (GT4) programmers [15]. It is built on top of Eclipse [17] which is an open-source IDE written in Java. Eclipse has an extensible plug-in system that allows it to be augmented with new features. It supports multiple programming languages and runs on multiple platforms where the Java Virtual Machine is available. It has the traditional IDE user-interface with multi-window editors, syntax highlighting support, and other features commonly found in advanced IDEs. GT4IDE targets the development of Grid applications as services specifically Globus Toolkit 4 services. It provides automatic generation of WSDL (Web Services Description Language) files and build tools 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 from Asia Pacific Science & Technology Center is a Grid-enabled IDE using NetBeans as the base IDE [16]. NetBeans [18] is an open-source IDE from Sun Microsystems. Similar to Eclipse, it has standard features commonly found in traditional IDEs, syntax highlighting, refactoring support, visual editors, and an extensible plug-in system. GriDE provides a traditional programming environment through NetBeans, a simple Grid workflow editor, a job submission tool, a resource browser, 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 resources on the Grid using the resource browser and with GriDE's job submission tool users can submit Globus Toolkit 2 jobs to the resources directly from the IDE. Some advanced functions like Grid debugging and complex Grid workflows are still unimplemented.

The existing tools described above benefit Grid programmers by providing development environments that ease and simplify the process of Grid application development. However 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 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. This abstraction allows programmers to write Grid applications with various IDEs and middleware. The framework, Worqbench, provides services and plug-ins for traditional IDEs that enable them to be used for Grid application development. It also provides

middleware adapters that allow users to target various Grid resources. Further motivation and objectives are discussed in the following section.

3. Motivation and Objectives

There are several non-trivial issues faced by e-Scientists writing Grid applications that motivate our research. These issues include: scalability, heterogeneity, and lack of standards.

Scalability. There are many aspects of scalability such as performance, fault tolerance, load management, and code maintainability. However, in this research we focus on the scalability of the application development environment. Specifically, we are concerned with the ability of the development process to scale as the number of Grid resources increases. As stated in the introduction section, Grid developers typically still write the applications on the local machine and then transfer them to Grid nodes to be executed. To debug the applications, the developers login using SSH to the execution nodes and run a debugger to attach to the local processes. This software development process does not scale as the number of execution nodes increases because it is time consuming, ineffective, and error prone.

Heterogeneity. Although Grid middleware defines a single set of common services for security, information, and management and offers a uniform way for executing Grid applications, it does not solve the problem of architectural 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 writing applications. There are two ways to approach this heterogeneity problem. One way is to hide the underlying differences by writing the applications in high-level languages that have portable interpreters and virtual machines (such as Python and Java). This approach may not fully exploit the advantages offered by some specific architectures and in practice, the majority of Grid programmers still write the applications in legacy languages such as C/C++ or Fortran [1]. Another way to approach the heterogeneity problem is to expose the differences but provide tools that assist developers. For example, many C programmers use conditional compilations such as `ifdef` to support multiple architectures. However, these conditionals cannot be managed easily, adding or removing support for an architecture requires the developers to manually inspect all `ifdef` occurrences in the code. In spite of the importance, no software tools using the second approach currently exist that can help programmers to

write applications for heterogeneous Grid resources effectively.

Lack of standards. Standards and protocols have been proposed and defined in Grid computing for the execution management of Grid applications. However, no standard has been defined for the development of Grid based software. For example, there is no common protocol or technique for debugging or profiling Grid applications across multiple Grid resources. 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. We believe there is a need to have a common base or standard for these development tools.

Our objective is to develop an architecture and a prototype implementation of an integrated framework for Grid application development. This framework, Worqbench, defines a testbed, a collection of resources, as a first class object. Once the environment is informed about the testbed, its resources and their attributes, it can expose this in a controlled way to the user. This allows the framework to alleviate the scalability and heterogeneity issues. In addition, Worqbench defines common protocols and interfaces that serve as a base for Grid development tools. Specifically, the desirable features of Worqbench include the ability to:

- scale across a large number of resources.
- simplify the software development process on Grid resources
- ease Grid application development across multiple computer architectures
- allow developers to perform programming tasks on local and remote Grid resources
- accommodate different Grid middleware and IDEs

4. Architecture of the Framework

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. The left tier supports the user interface, the IDE and associated plug-ins. The middle tier implements the framework that coordinates the connection between the IDE and the Grid middleware. The right tier provides the underlying back-end structure which may consist of one or more resources managed by different Grid middleware.

This three-tier model ensures modularity and flexibility. It allows the aggregation of Grid resources with different architectures and middleware into a single Grid testbed accessible from the programmer's preferred IDE.

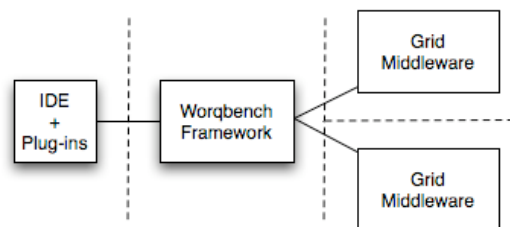


Figure 1. Three-tier model

Section 4.1 proposes Web Services as the underlying implementation of Worqbench. Section 4.2 gives the description of the components of the framework and section 4.3 describes its internal model. Section 4.4 discusses how Worqbench meets the challenges listed in section 3.

4.1. Web Services

Web Services are of special interest in Grid computing. They are platform and implementation independent software components and their function of interconnecting information systems is similar to the intended function of Grid computing [19]. The importance of Web Services is further strengthened with the adoption of Web Service Resource Framework (WSRF) as the standard for Grid middleware architecture [20][21].

Accordingly, Worqbench functions use Web Services. For example, we are implementing a Web Service for debugging, a Web Service for remote compilation and a Web Service for profiling program's execution time. These services can then be incorporated into an extensible IDE, providing programmers a familiar IDE user-interface as the front end, whilst using Web Services as the glue to the back-end Grid resources. The integration of these Web Services into an IDE is advantageous because it provides programmers a clear, easy, and seamless environment transition from traditional application programming to Grid programming. More importantly, these services do not replace those already provided in systems like Globus, but augment the existing standards specifically in the area of software development.

In addition, Worqbench provides a web interface that serves standard HTML pages with JavaScript. This web interface acts as a web portal and allows e-Scientists to access Worqbench services using a standard-compliant web browser. However, services which require a high degree of interactivity such as debugging are not accessible from a web browser. This type of service can only be utilized by calling Web Service methods using XML-RPC or SOAP protocols.

4.2. Components of the Framework

The architecture of the framework is shown in figure 2. The framework is composed of a number of key components, namely connection interfaces, application packager, development manager, database, building service, debugging service, profiling service, and middleware adapters.

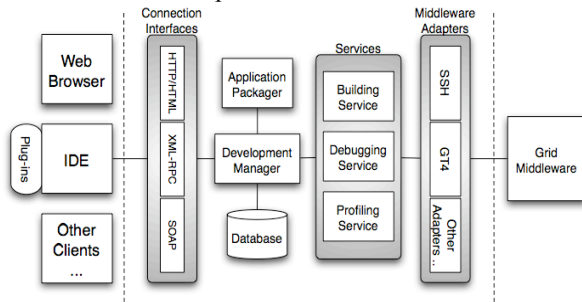


Figure 2. Key components in Worqbench

1. Connection Interfaces. The connection from the IDE and other clients to the framework is handled by the connection interfaces. Each interface handles one particular type of communication protocol. If an IDE requires a specific communication handler, a suitable connection interface can be written for that particular IDE without changing the core foundation. It is also possible, for example, to write a command-line interface client linked with a Web Service library that utilizes the framework through the XML-RPC or SOAP protocol.

2. Application Packager. Many IDEs have the concept of a project, where all of the necessary files to build an application such as header files, source files, and library files are grouped together into a project. IDE specific metadata files that describe the project are also stored in the project. The contents and the structure of the project are different between IDEs. In Worqbench, the heterogeneity of the IDE projects is not exposed to the Grid resources. Instead, Worqbench utilizes an application packager to convert an IDE specific project to an IDE independent project or package that can be built on the Grid resources without installing the IDE on the resources.

The application packager parses a project's metadata to obtain project specific information. The packager then creates a compressed file containing all of the necessary files and an instruction file to build the files independent of the IDE. A portion of the application packager is IDE specific since it needs to understand the metadata peculiar to the IDE.

3. Development Manager. The development manager is the core of the framework. It provides the linkage between the user and the IDE with the Grid resources. To represent the linkage and the state of

entities such as users, projects, resources, and resource groups, the manager maintains an internal model of the system in the database. The manager coordinates user's access to various services: building, debugging, and profiling services. It also acts as a front end manager to the database.

4. Database. The database acts as a storage server for the framework. It stores project files for the development manager and the application packager. It is closely tied to the development manager since the data of the internal model is also kept in the database. The database can be run on another machine easily to lessen the load of the development machine if necessary.

5. Building Service. The building service is responsible for building or compiling Grid applications. It obtains an IDE independent project code from the database, transfers it to a particular Grid resource or some resources, and compiles the code on the machines. The building service utilizes the build script generated by the application packager from the project to compile the code. During compilation, output from the compiler is directed back to the IDE and the user. The service assumes that the target resources have a suitable environment, including compilers and libraries, to build the applications.

6. Debugging Service. The debugging service allows programmers to debug running applications. It acts as a proxy between a back end debugger running on Grid resources and a front end debugger of the IDE. It passes the debug input/output between the IDE and the back end debugger.

Unlike compiling an application, debugging an application is an interactive process. In the compilation phase, not much user interaction is required and it does not directly involve the Grid runtime environment. However, debugging an application requires user interaction and the ability of the user to control the execution of the debugged program. On the Grid, where it is common to employ job schedulers or underlying Grid execution tools to manage the execution of applications, the user does not have precise control of the execution of the debugged program. This hinders the user to perform debugging operations on applications running on Grid resources.

To solve this problem, Worqbench utilizes a callback mechanism initiated by the back end debugger. When the job scheduler runs the back end debugger, it contacts and notifies the IDE. The debugger then stops the application and waits for a connection and user interaction from the IDE. With this technique, the user will be notified when the application has been started by the job scheduler and it allows the user to connect to the waiting application and to start the debugging operation.

7. Profiling Service. The profiling service enables the user to profile the executing Grid applications. Profiling an application requires little or no interaction from the user, however, it may generate statistical output. This output is forwarded by the service to the IDE where it can be rendered for display. Plug-ins for the IDE can be written to visualize the output from the profiling service. A similar callback mechanism to the debugging service is used by the profiling service.

8. Middleware Adapters. The middleware adapters are the glue between the framework and the low level Grid middleware. They translate generic input commands from the framework to specific Grid middleware commands or APIs. A generic method is provided to allow programmers to specify the API name to be called. Each adapter handles one particular Grid middleware platform. If a programmer uses resources handled by a new Grid middleware, a suitable adapter can be written for that particular middleware without changing the core foundation of the framework. Thus, supporting a middleware will be as easy as writing an adapter for the framework. The middleware adapters use the authentication information provided by the user such as X.509 proxy certificates or user passwords to gain access to the Grid resources.

4.3. Internal Model

Worqbench maintains an internal model of the system and stores the state of entities such as users, projects, resources, and resource sets. A conceptual entity-relationship diagram of the model is shown in figure 3. There are 6 model classes which are described below.

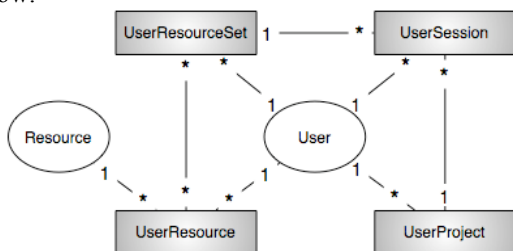


Figure 3. Internal model of the framework

Resource. It represents one distinct machine/resource in the framework. The class keeps a general information of a resource such as name, type, host address, and description. The respective class attributes are *resourcename*, *resourcetype*, *resourceaddr*, and *resourcedesc*. The Resource class does not hold user-specific information such as user name and user password. This information is saved in the UserResource class.

User. The framework revolves around the User class. It represents one user of the framework. It can be

a programmer using an IDE or a scientist accessing the framework via a web browser. As the figure shows, one user can have many UserResources, UserResourceSets, UserProjects, and UserSessions. The attributes in the class, *login* and *password*, store the authentication information of the user. All of the other classes: UserResource, UserProject, UserResourceSet, UserSession, have a reference to a particular User.

UserResource. It represents one machine/resource where the user has access to. The class keeps a reference to a Resource object. The UserResource fields are: *resname*, *resusername*, *resuserpass*, and *resauthfile*. A user given name of the machine which can be distinct from *resourcename* in Resource is stored in *resname*. The account user name for the machine is stored in the *resusername* field. The *resuserpass* field keeps a password string for resources that rely on a username-password pair as the authentication mechanism. Others such as Globus Toolkit 4 resources use a proxy certificate file to authenticate the user. The *resauthfile* field holds this authentication file of the user for the framework. UserResources can be combined to form a testbed or a resource set.

UserProject. The application development project for the user is represented by the UserProject class which has two fields: *projectname* and *projectdata*. The *projectname* field holds the name of the project while the *projectdata* stores a zip file containing all necessary files for the development project. It is essentially a compressed file of the project directory. The model allows a user to maintain multiple independent projects.

UserResourceSet. The user may want to group some resources to form a testbed or a collection of resources. This testbed is represented by the UserResourceSet class which has a single field *setname* that stores the name of the resource set. The class maintains a collection of references to various UserResource objects.

UserSession. A UserSession class represents a development session for the user. It is defined as a programming session where the user develops an application in one UserProject to be tested and executed on one UserResourceSet. The class has one field, *sessionname*, that keeps the name of the session. The UserSession class has references to one UserProject and one UserResourceSet.

Worqbench services such as building, debugging, and profiling services require one UserSession object as one of the parameters. An example of a client calling various methods in Worqbench is given in section 5.

4.4. Meeting the Challenges

Worqbench addresses the challenges listed in section 3, namely scalability, heterogeneity, and standards.

Scalability. Worqbench scales to large testbeds because it performs development tasks on *behalf* of a user rather than requiring the user to perform these manually. For example, the building service copies files from the user's project and builds the project on each resource without user interaction. This helps the user manage a software development project across a large number of target resources, as experienced in the Grid. The scalability issue is also alleviated by the development manager that handles the testbeds on *behalf* of the user. In addition, multiple Worqbench servers can be used by a single IDE to increase the scalability.

Heterogeneity. Worqbench exposes heterogeneity to the user in a controlled way and therefore helps manage the challenge in the Grid. For example, the building service provides Grid resource information to the application packager. It checks whether a group of resources is sufficiently homogenous so that only one project build is required. If they are similar, the application packager then presents them as one type of resource to the user. It helps the user manage the variability in platforms by linking source code development techniques (such as conditional compilations) to the composition of the testbed itself.

Lack of standards. Worqbench proposes and implements standard methods that could support a number of tools and projects for Grid application development. Worqbench provides unified services for building, debugging, and profiling Grid applications. Whilst these augment existing middleware, a range of adapters is also provided that allows the user to work with various middleware stacks.

5. Implementation Details

Worqbench is written in Ruby [22] and is built on top of Rails. Rails is a full-stack framework for developing database-backed web applications according to the Model-View-Controller (MVC) pattern [23]. 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 [24]. It is easily extended and it has a low system requirement.

5.1. Eclipse IDE

To date, in this project, plug-ins for the Eclipse IDE and adapters for SSH and Globus Toolkit 4 have been developed. These allow users to manage remote SSH and GT4 resources and to compile and to execute Grid applications on these resources from Eclipse as depicted in figure 4. These plug-ins differ from various Eclipse plug-ins that are available for the development of Web Services and Grid services (GT4IDE). We believe the development of traditional Grid applications is still important in the foreseeable future as the development of Grid services increases and matures.

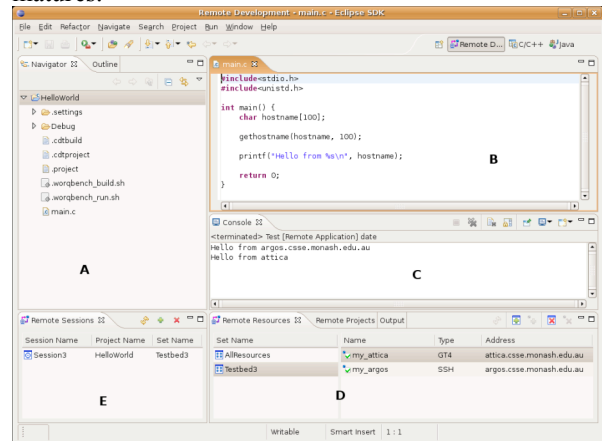


Figure 4. Eclipse IDE

Figure 4 shows an Eclipse IDE with five windows. Window A shows files and folders in an Eclipse project named HelloWorld. The project also contains Eclipse specific metadata files, for example, .cdtbuild, .cdtproject, and .project. Window B shows the source code editor. Windows C, D, and E are our extensions to the IDE. Window C displays the output from remote resources. Window D shows the resource browser with two panes. The left pane displays resource sets that the user has created (Testbed3) and special resource sets such as AllResources that shows all available resources. The right pane displays resources in the currently selected resource set. Window E shows the Worqbench sessions of the Eclipse user. The current session, Session3, indicates that the Testbed3 resource set is used as the development testbed for the HelloWorld project.

To run the remote application, the user uses the standard Eclipse configurations dialog. The output of the executed application is displayed in the output window (window C in figure 4). In the figure, it shows the output of main.c program which has been executed on two resources, my_attica (GT4) and my_argos (SSH).

5.2. Python Client

Since Worqbench's functionalities are exposed as Web Services, they are readily available for clients written in other languages. The following Python code listing shows an example of utilizing Worqbench APIs.

```
01 import xmlrpclib
02 server = xmlrpclib.Server
('http://localhost:3000/service/api')
03 key = server.admin.GetKey
('username', 'password')
04 server.resource.New(key,
'machine1', 'login', 'pass', ...)
05 machine1_id = server.resource.Get
(key, 'machine1')['id']
06 server.set.New(key, 'testbed')
07 server.set.AddResource
(key, 'testbed', machine1_id)
```

6. Summary and Conclusion

This paper describes the motivation and design of Worqbench, an integrated framework for Grid application development. Worqbench acts as an intermediary between existing IDEs and Grid middleware, allowing Grid programmers to gain the substantial benefits of using an IDE. It is designed with a layered architecture to make it modular and flexible to accommodate different IDEs and different Grid middleware. It provides adapters that allow programmers to work with Grid resources managed by different middleware. Worqbench provides services for application development such as compiling, debugging, and profiling services that can be used by traditional IDEs through appropriate plug-ins. These Web Services also enable Worqbench to be accessible by other software components that are implemented on different platform and in different languages than the framework.

Worqbench, however, is still under active development. A debugging service support for debugging GT4 applications has been written although it still needs to be further integrated into the Eclipse IDE. Preliminary testing of Worqbench using Eclipse and SSH & GT4 adapters has been performed. Further testing using various Grid middleware adapters and Grid resources need to be conducted to measure how well Worqbench achieves the objectives described.

7. References

[1] Balle, S. M. and Hood, R. T., GGF UPDT User Development Tools Survey, March 2004.

[2] National e-Science Centre, Definition of e-Science, <http://www.nesc.ac.uk/nesc/define.html>.

[3] Foster, I., Kesselman, C. and Tuecke, S. (2001) The anatomy of the grid: enabling scalable virtual organizations. *International Journal of Supercomputer Applications*.

[4] Joseph, J. and Fellenstein, C., *Grid Computing*, Prentice Hall, 2004.

[5] GDB - <http://www.gnu.org/software/gdb/gdb.html>.

[6] Reis, C. and Cartwright, R. (2004) Taming a Professional IDE for the Classroom. *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*.

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

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

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

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

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

[12] Kacsuk, P., Dozsa, G., Kovacs, J., Lovas, R., Podhorszki, N., Balaton, Z., Gombas, G., P-GRADE: A Grid Programming Environment, *Journal of Grid Computing*, Volume 1, Issue 2, 2003, Pages 171 - 197

[13] Berman, F., Chien, A., et al. The GrADS Project: Software support for high-level Grid application development. *International Journal of Supercomputer Applications* 15, 4 (2001), 327 - 344.

[14] Huang, R., Casanova, H., and Chien, A (2006) Using Virtual Grids to Simplify Application Scheduling, *IEEE International Parallel and Distributed Processing Symposium*, 2006.

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

[16] GriDE - <https://gride.dev.java.net/>.

[17] Eclipse - <http://www.eclipse.org/>.

[18] NetBeans - <http://www.netbeans.org/>.

[19] Atkinson, M. Rationale for choosing the Open Grid Services Architecture, in *Grid Computing: Making the Global Infrastructure a Reality*, Wiley, 2003.

[20] 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.

[21] Czajkowski, K., Foster, I., Frey, J., et al. (2004) The WS-Resource Framework, Version 1.0. <http://www.globus.org/wsrf/>.

[22] Ruby - <http://www.ruby-lang.org/>.

[23] Ruby on Rails - <http://www.rubyonrails.org/>.

[24] Ruby on Rails and J2EE: Is there room for both?, <http://www-128.ibm.com/developerworks/linux/library/wa-rubyonrails/?ca=dgr-lnxw16RubyAndJ2EE>.