

# COMPONENT-BASED DEVELOPMENT ENVIRONMENT FOR GRID SYSTEMS: DESIGN AND IMPLEMENTATION \*

Artie Basukoski, Vladimir Getov, Jeyarajan Thiyagalingam, Stavros Isaiadis  
*Harrow School of Computer Science*  
*University of Westminster*  
*Watford Road, Northwick Park*  
*Harrow, London HA1 3TP, U.K.*

A.Basukoski02@westminster.ac.uk

V.S.Getov@westminster.ac.uk

T.Jeyarajan@westminster.ac.uk

S.Isaiadis@westminster.ac.uk

**Abstract** Component-oriented development is a software design method which enables users to build large scale Grid systems by integrating independent and possibly distributed software modules (components), via well defined interfaces, into higher level components. The main benefit from such an approach is improved productivity. Firstly, due to abstracting away network level functionalities, thus reducing the technical demands on the developer. Secondly, by combining components into higher level components, component libraries can be built up incrementally and made available for reuse. In this paper, we share our initial experiences in designing and developing an integrated development environment for Grids to support component-oriented development, deployment, monitoring, and steering of large-scale Grid applications. The development platform, which is tightly integrated with Eclipse software framework, was designed to empower the developer with all the tools necessary to compose, deploy, monitor, and steer Grid applications. We also discuss the overall functionality, design aspects, and initial implementation issues.

**Keywords:** Component-oriented development, Grid platform, integrated development environment, Eclipse.

\*This research work has been partially supported by the GridCOMP and CoreGRID projects funded by the European Commission.

## 1. Introduction

Grid systems have become tightly integrated as an indispensable part of the computing community for solving problems in different domains. Computational Grids offer remarkable benefits for solving a given problem, especially in connection with performance and resources. Although the main focus is about the runtime performance, the actual investment in terms of time includes both the execution time and the time for software development. This implies that the development experience has a direct impact on the “time-to-solution” cycle.

The software development can be simplified by following a component-oriented paradigm [2], where faster development is achieved through higher levels of abstraction. Although there exists substantial amount of ground work in facilitating the design and the utilisation of modern Grid systems, rarely do any of them offer a unified and integrated solution with the support of full-fledged component-oriented development. It has been recognized, however, that component technologies and their associated tools become very attractive for building complex Grid applications [8]. Indeed, when constructing distributed programs from separate software components, the ability for rapid composition and the support for dynamic properties at runtime may bring substantial reductions to the “time-to-solution” cycle.

In this paper, we discuss our initial experiences in developing a component-based Grid integrated development environment (GIDE). The environment is designed based on the model-driven approach using standard software tools for both development and integration within the Eclipse framework [4]. This essentially means that the platform will be hosted as part of the Eclipse framework, enabling developers to leverage the benefits of both the Eclipse and GIDE environments.

Our Grid IDE offers extensive support for component-oriented development and for post-development functionalities covering deployment, monitoring, and steering. In essence, the environment offers full support for different user groups of Grid – developers, application users and data-centre operators. In supporting the development, we embrace the Grid Component Model (GCM) [14], which, unlike other component models, truly supports various aspects of Grids, in terms of programming – heterogeneity and dynamicity.

The paper is organized as follows: Section 2 provides an overview of related work while Section 3 discusses requirements arising from different user groups. The architectural design of the GIDE is presented in Section 4. Finally, Section 5 concludes outlining some directions for future work.

## 2. Related Work

Providing support for Grid applications has been a major focus of many different projects. However, notable differences exist based on the target user group being addressed.

Friese et al. [6] discuss a set of Eclipse-based development tools for Grids, based on the model-driven approach. Their approach is to support the development through Unified Modelling by providing a well separated model mapping layer. In essence, their approach relies on two different layers where the top-level layer provides the model information while the underlying layer provides the correct mapping for the underlying platform. Their tool set covers the automation of this mapping between the layers. Once developed through the appropriate model, applications can be monitored, deployed and maintained through the tool set. This is in contrast to our approach, where we rely on a well-specified Grid component model and a strict software engineering approach. Further, although both their work and our work rely on the Eclipse platform for providing rich functionality, our work is more user friendly and more developer-oriented through permitting visual composition of applications. However, the target user group covers the same group as ours.

The Grid Engine (GridE) and sub projects thereof from Sun Microsystems [15] provide substantial support for development. This, however, in contrast to our work, is entirely targeted towards work-flow based applications without any explicit notion of components. The underlying platform is Netbeans [11].

The Web Tools Platform project [16] of Eclipse also aims at providing support for Grid applications from within Eclipse. However, the main support is through wizards for creating Java web services or variants without any clear support for Grid-specific issues.

The g-eclipse [7] project provides an integrated workbench in a similar manner to us. Its main focus is on what we refer to as steering and provides tools for monitoring Grid resources, job creation, deployment and inspection. There is however, no support for graphical composition.

In addition to all these, there are other projects that [9, 3] offer support for developing Grid applications. However, it is not uncommon to see that the majority of them do not offer well-specified and clear support for any Grid-specific component models.

## 3. An Overview of Requirements

The Grid IDE is aimed at supporting a number of different user groups. We can classify the user groups as follows:

*Application Developers:* Application developers require support for developing Grid applications through graphical composition as well as having to support

source-code based development. This approach aligns with industrial efforts in building applications through graphical composition [4]. However, providing support for developing Grid applications poses additional requirements, including support for Grid component models and composite components, and the complexities of deploying these components over distributed systems. Additional tools are necessary to enable deployment, monitoring of both component status and resource, and steering of components to maximise resource utilisation.

*Application Users:* The GIDE should facilitate the deployment of applications and subsequently, the monitoring of deployed applications. The monitoring process provides a set of opportunities for concerned users to monitor their running application in real-time. This functionality is also shared with the application developers who need such facilities to test the application during development.

*Data Centre Operators:* Data centres have high turnover rates. Hence there is a need for a design that would facilitate fast handovers and enable other operators to assist newcomers in coming to terms with the applications quickly. In order to achieve this we intend to deliver a standalone application as a Rich Client Platform (RCP) application, which provides the key functionalities that would be required by a data centre. These features are arranged within the deployment, monitoring, and steering perspectives. Also, personalisation of views should be limited as far as possible, so that a uniform design is visible to all operators in order to enhance handover and communication.

The following is an overview of the key requirements considered for each user group during the design of the GIDE.

- 1 Provide a Grid IDE for programmers and composers.

The main goal is to produce an integrated programming and composing GUI. It should provide the developer with graphical tools to develop both normal code and legacy code into primitive components, as well as tools for assembling existing Grid components into larger composite components. Additional support tools should also be provided, such as tools to search for suitable components, and tools to finalise the configuration of the application before execution.

- 2 Provide tools for the deployment of a given Grid component configuration or application.

The main goal is to develop a component launcher tool that enables the developer to simply point to a component and execute. Of course the launcher will need to associate a deployment descriptor with each launched component. In addition this tool must provide monitoring at execution. This can be achieved via a components execution monitor

tool, capable of monitoring the runtime dynamics of set of components, such as location, memory, status, etc.

### 3 Provide a Grid IDE for data-centre operators.

A simplified tool for installing, monitoring and mapping necessary component code to available resources. The tool must support steering, for installing, removing, and re-installing new versions of component code. It must also provide tools for the monitoring of resources. These include usage level of resources required for execution of component based code, as well as external services the components might need to execute.

## 4. Design of the GIDE

Our vision for the GIDE design is to provide the developer with a single environment for the development pipeline. As can be seen from Figure 1 this includes graphical composition, deployment, monitoring and steering of grid based applications.

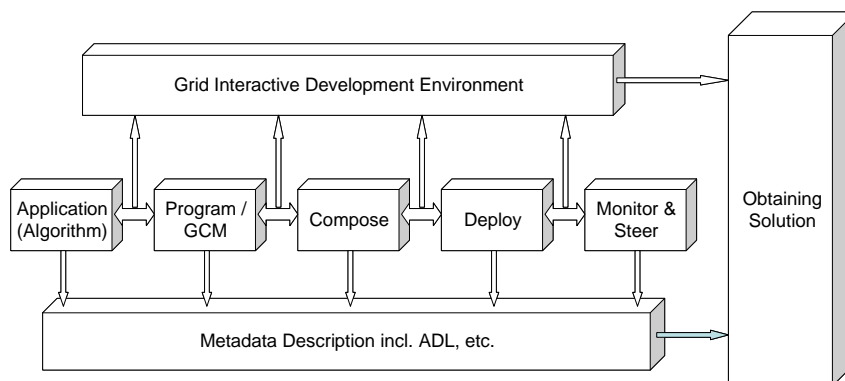


Figure 1. Component-Based Program Development Pipeline

Our philosophy is to restrict the programmer as little as possible, and enable the developer full access to all levels of the language hierarchy. By language hierarchy we mean that the developer will be capable of switching between developing graphically via a Component Model view, and coding directly in a suitable computer language using some middleware API. Given that the underlying component model for our platform is GCM we selected Java and Eclipse as the development platform. This enables the maximum integration with the ProActive library [12], which is the GRID middleware for this reference implementation. Eclipse is also well known for its extensibility via the development

of suitable plugins and hence provides a seamless path from code to component model in a uniform development platform. This ensures that the target user groups can benefit from a richer set of functionalities [6].

In addition to this, deployment, monitoring and steering are also being developed as plug-ins. Some monitoring capability is already present in the Interactive Control and Debugging of Distribution (IC2D) application [1] which provides graphical monitoring of Active Objects. This needs to be extended in order to enable the deployment and monitoring of components. The main advantage of relying on this plug-in-based approach is that specific features could be activated (plugged-in) on demand. Figure 2 gives a block diagram representation of the GIDE design.

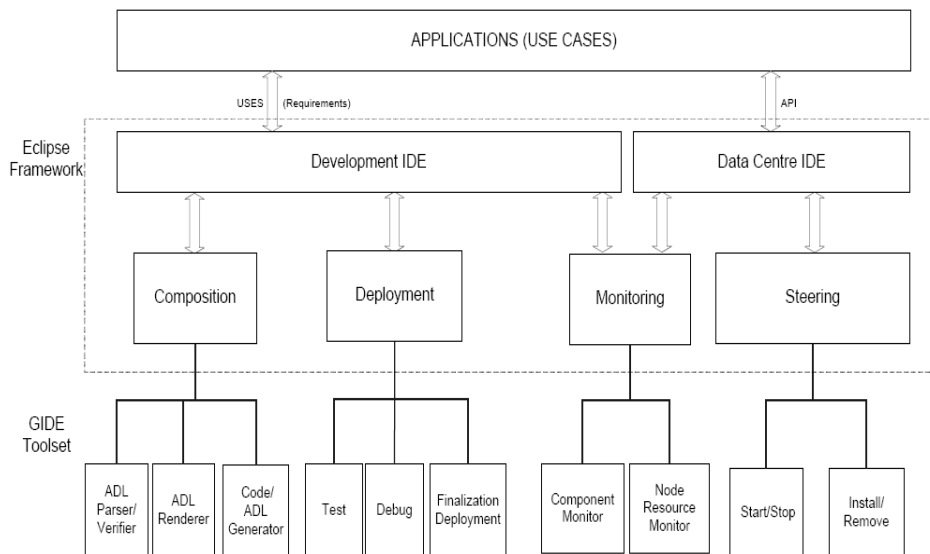


Figure 2. GIDE Block Diagram

In designing the front-end, we followed the model driven approach supported by the Eclipse platform. In the case of composition, our model is the final output from the IDE — composition. The underlying architecture of the IDE relies on this model for the functionalities. The model is well supported by a front end based on the Graphical Editing Framework (GEF) [5] and inherited features from GEF, such as event handlers.

## 4.1 Composition

The composition process is enabled via a fully interactive environment. The underpinning feature which enables such interactivity is the event driven approach. Eclipse acts as the host platform to our environment. The Graphical Editing Framework, GMF-Runtime, and Eclipse facilitate handling of different events within the environment. These events are captured by the host platform through a message loop processed by the Eclipse, which are then routed to the dedicated event handlers or providers of the environment. These event handlers or providers are linked to the underlying model so that the changes are reflected upon editing

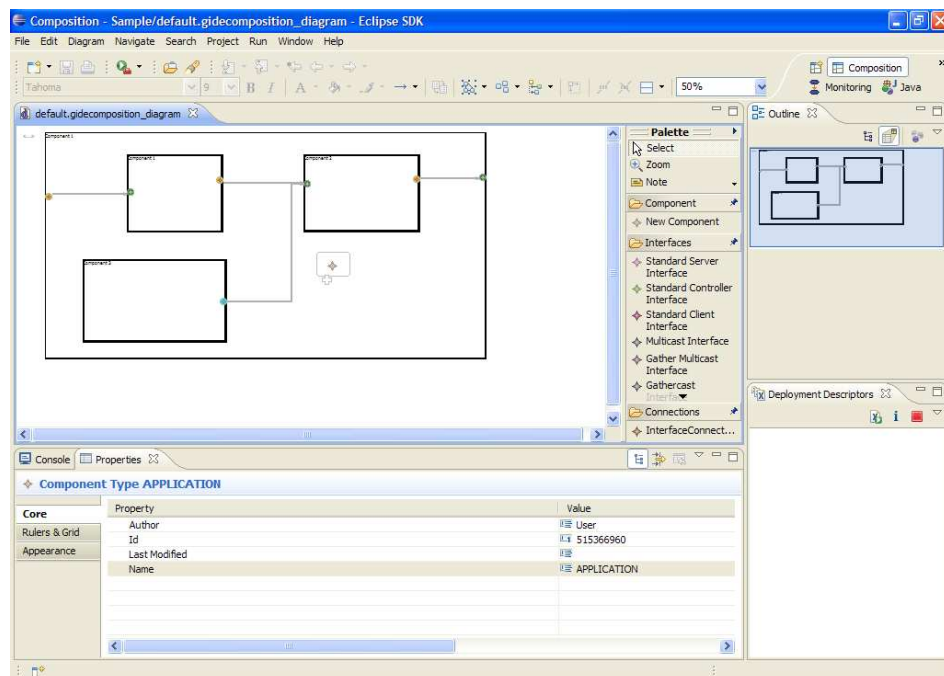


Figure 3. Component Composition Perspective

A prototype has been completed for the Composition perspective (see Figure 3). The central area focuses the user on the graphical composition view which provides the developer with a palette of available components that can be dragged and dropped on the composition canvas. Components can also be imported from existing Architecture Description Language (ADL) files and stored in the component palette. ADL files conform to the GCM-specification for describing compositions such as in [13]. Components can then be resized

and moved, modified and stored. Connections between the interfaces can be drawn directly between the components using the connection tool. Composition is achieved by drawing a membrane around a group of components and defining interfaces. The developer is able to switch between the graphical view, and a view of the fractal description of the component as an ADL file. The ADL file can then be exported and used for deployment.

## **4.2 Deployment Perspective**

This perspective consists of views needed for application deployment. The main view is of a deployment descriptor editor to map physical hosts to virtual nodes. Deployment descriptors are used to associate components with virtual nodes. Virtual nodes are included in ADL files to specify the number of virtual nodes that the component will need. A developer may have a set of these deployment descriptors to be used for deployment to different hardware configurations. To complement this view, a view of the hosts and their resource statuses is also provided, giving a developer the ability to associate sets of hosts with each deployment descriptor. Within the deployment perspective the operator is able to launch components simply via drag-and-drop operations before moving on to steering.

## **4.3 Monitoring Perspective**

The monitoring perspective provides the views that data centre operators need in order to properly monitor the environment in which components operate. See Figure 4 for an example Monitor perspective consisting of component and resource monitor views. Three types of monitoring are necessary in order to enable proper management of applications. Firstly, monitoring of resources provides the hardware status of hosts. This includes CPU utilization, hard disk space, and other platform specific status information. Secondly, monitoring of the GCM components themselves provides status and location information along with a zoom-in feature for monitoring sub-components. Finally, we allow monitoring of active objects, which is necessary for developers/composers to debug and monitor applications during the development phase.

## **4.4 Steering Perspective**

More useful for data centre operators, the aim of the steering perspective is to provide views to enable the operator to start, stop and relocate components. Building on the monitoring and host views, it has as its main focus a component monitoring view. This view graphically shows the components location and their status. An additional view shows the geography and resource availability of the hosts, virtual nodes, as well as the components that are running on them. Based on these views, the operator has the facility to start, stop or move

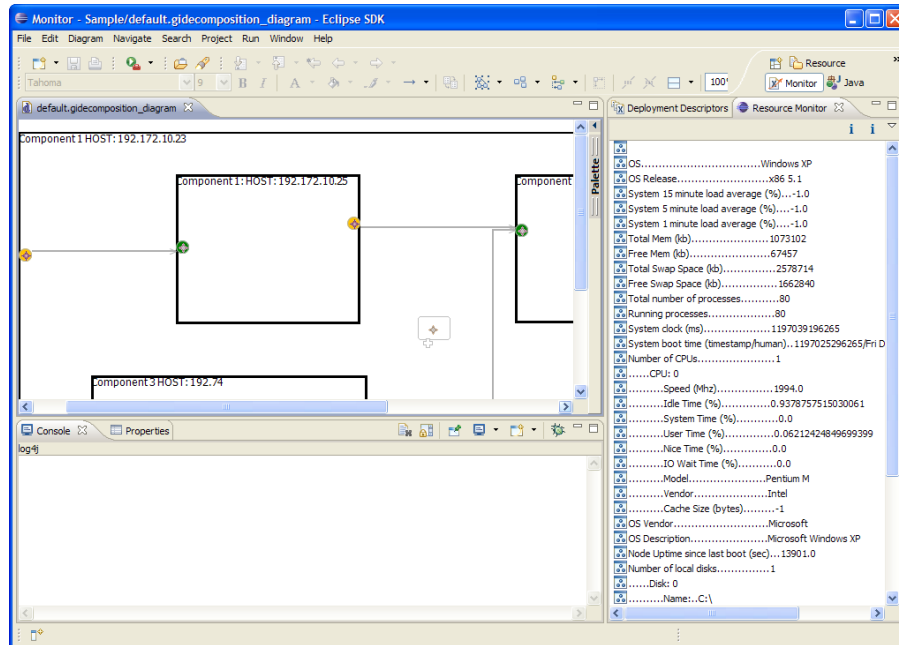


Figure 4. Monitor Perspective

components from one virtual node to another while monitoring their status to ensure correct execution.

## 5. Conclusions and Future Work

We have outlined our approach in designing and implementing a component-based development environment for Grid, targeting different user groups. Our environment utilises some existing work and is based on the Eclipse framework. The environment follows both the model- and event-driven approaches and offers better support for different user groups. We have implemented a prototype where the underlying component model is GCM. The prototype provides support for composition and monitoring of component-based applications. The IDE provides seamless support for both high level graphical composition, and low level source code access of the resulting compositions. This approach facilitates debugging, and does not restrict advanced users by forcing them to solve

all issues via composition for cases where it may not be the most appropriate solution.

Future work:

- While the GIDE currently supports the automatic generation of ADL files through an export facility, generating Java skeleton source files for components is an essential feature.
- There are plans to include a live composition validation feature, which will inform the developer when there is an error in the current composition through non-intrusive visual means (e.g. a red/green flag on the Composition perspective).
- Further to host monitoring, another important feature is the runtime monitoring of components, for instance, component queue status, load, open connections, etc. We intend to provide some support for such tasks in a feature version.
- Wherever applicable, especially in the case of ADL files, we intend to offer context-specific syntax highlighting within the source code.

## References

- [1] F. Baude, A. Bergel, D. Caromel, F. Huet, O. Nano, and J. Vayssière. IC2D: Interactive Control and Debugging of Distribution, Proc. Int. Conference on Large-Scale Scientific Computing, Lecture Notes in Computer Science, Vol. 2179:193–200, 2001.
- [2] C. Szyperski. Component Software: Beyond Object-Oriented Programming, 2nd ed. Addison-Wesley, 2002.
- [3] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. The International Journal of High Performance Computing Applications, Vol. 15(4):327–344, 2001.
- [4] Eclipse - An open development platform. <http://www.eclipse.org>.
- [5] Eclipse Graphical Editing Framework. <http://www.eclipse.org/gef/>
- [6] T. Friese, M. Smith, and B. Freisleben. Grid Development Tools for Eclipse. Eclipse Technology eXchange Workshop eTX at ECOOP. 2006.
- [7] g-Eclipse Project. <http://www.geclipse.org/>
- [8] V. Getov, G. von Laszewski, M. Philippsen, and I. Foster. Multiparadigm Communications in Java for Grid Computing. Communications of the ACM, Vol. 44(10):118–125, 2001.
- [9] GMT Project. <http://www.eclipse.org/gmt/>
- [10] Model Driven Architecture. <http://www.omg.org/mda>.
- [11] Netbeans IDE. <http://www.netbeans.org/>.
- [12] ProActive Java Grid Middleware. <http://www-sop.inria.fr/oasis/proactive/>
- [13] The Fractal Project. <http://fractal.objectweb.org/>

- [14] Proposal for a Grid Component Model. CoreGRID Deliverable, D.PM.002, 2005.
- [15] S. See, J. Song, L. Peng, A. Stoelwinder, and H.K. Neo. GriDE: A Grid-Enabled Development Environment, Proc. Int. Workshop on Grid and Cooperative Computing, Part I, Lecture Notes in Computer Science, Vol. 3032:495–502, 2003.
- [16] Web Tools Platform Project. <http://www.eclipse.org/webtools/>