Integrated framework for development and execution of component-based Grid applications

Vladimir Getov

Harrow School of Computer Science

Integrated Framework for Development and Execution of Component-based Grid Applications

Vladimir Getov
Harrow School of Computer Science, University of Westminster,
Watford Rd, Northwick Park, Harrow, London, HA1 3TP, U.K.
V.S.Getov@westminster.ac.uk

Abstract

Component-based software technologies have emerged as a modern approach to software development for distributed and Grid applications. However, the lack of longer-term experience and the complexity of the target systems demand more research results in the field. This paper provides an overview of three different approaches to developing component-based Grid applications. In order to re-use legacy codes, the wrapper software approach can be adopted in its two flavours – hand-written or automatically generated wrapper code. Another approach applicable to existing object-oriented software is to componentise the code by introducing appropriate modifications. The third approach is component-oriented development from scratch. We compare and contrast the three approaches and highlight their advantages and weaknesses.

1. Introduction

It is generally accepted that component-based software development is becoming the most cost-effective approach to application construction for complex distributed and Grid systems. This, however, makes the search for the most appropriate programming model and corresponding programming environments even more important than before. Arguably the most serious obstacle to the acceptance of modern component technologies is the so-called *software crisis*. Software, in general, is considered the most complex artefact in distributed computing; since the lifespan of Grid infrastructures has been so brief, their software environments rarely reach maturity making the software crisis especially acute. Hence, portability and support for dynamic properties, in particular, are critical issues in enabling large Grid computing systems.

The wide adoption of component-based software development and in particular the use of suitable programming models for compatibility and interoperability are key issues towards building effective future Grids. Examples of component models applicable to this field include the Common Component Architecture (CCA) [1], the CORBA Component Model (CCM) [6], and the emerging Grid Component Model (GCM) [4]. The main aim of this work is to present our experience in applying component-based development using different approaches depending on the status and the properties of different application codes.

The rest of this paper is structured as follows. Section 2 provides background about the major component models in the field. Section 3 presents the wrapper generation approach for re-using legacy codes. Section 4 describes the second approach to reengineering existing object-oriented applications, which comprises a general componentisation process. Section 5 presents the third approach for developing component-oriented application codes from scratch. Finally, Section 6 concludes the paper.

2. Background

The Fractal specification [3] proposes a generic, typed component model in which components are runtime entities that communicate exclusively through interfaces. One of the crucial features of this model is its support for hierarchical composition. Another key feature is its support for extensible reflective facilities: each component is associated with an extensible set of controllers that enable inspecting and modifying internal features of the component. Controllers provide a means for capturing extra-functional behaviours such as varying the sub-components of a composite component dynamically or intercepting incoming and outgoing operation invocations. The GCM proposal [4]

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

is an extension of the Fractal component model that specifically targets Grid environments.

The Common Component Architecture (CCA) [1] specifies the means for interaction among components. In CCA, components interact using ports, which are interfaces pointing to method invocations. Components in this model define *provides-ports* to provide interfaces and *uses-ports* to make use of non-local interfaces. The enclosing framework provides support services such as connection pooling, reference allocation and other relevant services. Dynamic construction and destruction of component instances is also supported along with local and non-local binding. Though CCA enables seamless runtime interoperability between components, one of the main weaknesses of the CCA is the lack of support for hierarchical component composition and for control mechanisms thereof.

The CCM [6] is a language-independent, server-side component model which defines features and services to enable application developers to build, deploy and manage components to integrate with other CORBA services. The CCM is an extension of the CORBA object model defined to overcome its complexities. The CCM specification introduces the concept of components and the definition of a comprehensive set of interfaces and techniques for specifying implementation, packaging, and deployment of components. The CCM provides the capabilities for composing components (through receptacles) and permits configuration through attributes. However, in contrast to the Fractal component model, the CCM does not permit hierarchical composition; that is, recursively composing components to form more complex, composite components.

3. Wrapping legacy software

Giving high level of attention and support to legacy applications by providing enabling approaches and tools for their seamless integration into state-of-the-art component-oriented systems is a high priority issue. Depending on the properties and the development status of different legacy codes two approaches can be considered – automatically generated wrapping [5], and hand-coded wrapping [8]. In both cases the wrapper development process can significantly be simplified by adopting the most efficient strategy and using various tools for reducing the time for development. An important aspect of this strategy is to guarantee relatively small overhead introduced by the wrapper software layer.

4. Componentizing existing applications

This approach consists of a general componentisation process in order to transform an object-based system into a component-based system. The process assumes that the target component platform allows connecting components via provided and required interfaces, and that it minimally supports the same communication styles as the object platform (e.g., remote method invocation, streams, and events) [7].

5. Developing component-based codes

The development process for new application codes can be simplified by following the component-oriented development paradigm [2]. This approach normally involves the use of a component-based Grid integrated development environment (GIDE), which supports component-oriented development and post-development functionalities such as deployment, monitoring and steering. These functionalities target different user groups of the Grid – developers, application users and data-centre operators. Furthermore, the philosophy of the GIDE is to provide enhanced support with user-friendly graphical interface while enabling direct code editing. This means that a developer can freely switch between graphical development and direct coding of the required artefacts.

6. Conclusions

This paper provides an overview of three different approaches to developing component-based Grid applications. In order to re-use legacy codes, the wrapper software approach can be adopted in its two flavours – hand-written or automatically generated wrapper code. Another approach applicable to existing object-oriented software is to componentise the code by introducing appropriate modifications. The third approach is component-oriented development from scratch. We compare and contrast the three approaches and highlight their advantages and weaknesses. We present our experience in selecting and applying the most appropriate approach depending on the status and the properties of different application codes. The three main approaches can be integrated into a single development framework. The paper can also serve as a starting point for future developments in the area of component-based methodologies for constructing Grid applications.

Acknowledgement

This research work was carried out under the FP6 network of excellence CoreGRID (Contract IST-2002-004265) and the FP6 research and development project GridCOMP (Contract IST-2005-034442) funded by the European Commission.

References

[1] R. Armstrong *et al*, "Toward a Common Component Architecture for High-Performance Scientific Computing", *Proc. of IEEE HPDC Conference*, IEEE CS Press, 1999.

[2] A. Basukoski, V. Getov, and J. Thiyagalingam, "Component-oriented Development Environment for Grid: Design and Implementation", in *Making Grids Work*, Springer, 2008 (to appear).

[3] E. Bruneton, T. Coupaye, and J.B. Stefani, "Recursive and dynamic software composition with sharing", *Proc. 7th Int. Workshop on Component-Oriented Programming*, 2002.

[4] CoreGrid NoE – Institute on Programming Model, "Basic Features of the Grid Component Model", *Deliverable Report D.PM.04*, 2007.

[5] V. Getov, "A Mixed-Language Programming Methodology for High Performance Java Computing", in *The Architecture of Scientific Software*, Kluwer Academic Publishers, 2001, pp. 333-347.

[6] Object Management Group Inc, *The CORBA Component Model*, Revision V4.0, 2006, http://www.corba.org/.

[7] N. Parlavantzas, V. Getov, M. Morel, F. Baude, and D. Caromel, "Design Support for Componentising and Grid-enabling Scientific Applications", *Proc. ACM CompFrame'07 Symposium*, ACM Press, 2007, pp. 31-38.

[8] J. Thiyagalingam, V. Getov, S. Panagiotidi, O. Beckmann, and J. Darlington, "Domain-Specific Metadata for Model Validation and Performance Optimisation", In: *Achievements in European Research on Grid Systems*, Springer, 2007, pp. 165-178.