

Preface

I Abstract

Grid computing is an emerging technology that allows the users to conveniently access and share different kinds of resources – processing, data and networks – which are distributed worldwide over the Internet. Grids promise for broad user communities a transparent access to these resources, similar to today’s user-friendly access to electricity by plugging into the power grid. Grid technology is viewed as an enabler for both high-performance and data-intensive computing, and is therefore sometimes called the future Internet. The application fields of grid computing are manifold and include high-energy physics, crash test simulations, climate research, computational biology, etc.

A major challenge in grid computing remains the application software development for this new kind of infrastructure. Grid application programmers have to take into account several complicated aspects: distribution of data and computations, parallel computations between different sites and processors, heterogeneity of the involved computers, load balancing, etc. This makes grid programming a cumbersome task which might inhibit the acceptance of grids by a broad users community. The contemporary grid middleware systems – Globus and others – tackle some of these problems; however, they require from the programmer to explicitly provide a specific middleware setup using special formats and tools like WSDL and others. This middleware-specific setup must be typically done from scratch for every new application and completely rewritten if the application is changing, thus making the programmer’s task even more complex and reducing the software re-use. It is widely agreed that new programming models for grids are necessary which hide from the application software developer the details of the underlying grid infrastructure and middleware.

This book presents *Higher-Order Components (HOCs)* – a novel grid programming model, and puts it into a broader context of the contemporary distributed programming. HOCs are generic software components that can be parameterized not only with data but also with application-specific code (thus, the name higher-order). HOCs implement various patterns of parallel and distributed processing, such as farm, pipeline, divide-and-conquer and others. Implementations of components are provided to the programmer via a high-level, Internet-accessible service interface. A HOC implementation includes all necessary parallelization and synchronization and also the required middleware setup, which remain hidden from the programmer. The

programmer develops an application by composing suitable HOCs and customizing them for the particular use case by means of application-specific parameters (which may be either data or code). Thus, the application developer is concerned with the algorithm and other relevant aspects of the particular application, while the low-level concerns of the required middleware setup and the internal communication and synchronization within the employed parallel patterns remain hidden from him as a part of the HOC implementation.

The book describes the specific HOCs' features within the contemporary landscape of the component-based approaches in general and grid components in particular. We pay special attention to the Fractal component model which serves as a basis for the unified European Grid Component Model (GCM) which has been developed within the CoreGRID Network of Excellence. Both models – Fractal and GCM – provide a rich, well-founded set of composition opportunities which enable an efficient component-based development of complex applications. We describe how our higher-order components add to these features the possibility to use high-performance implementations of parallel processing patterns, and to employ standard Web services for exchanging data and code between HOCs and other application entities. The choice of Web services as the underlying communication technology makes HOCs interoperable with Fractal, GCM, and other contemporary distributed technologies.

In addition to general considerations, we present in detail the implementation approach taken for higher-order components. The book describes HOC-SA – the service architecture for HOCs – and its use as the runtime-system for component-based grid applications. We explain the relation between HOC-SA and the very popular Globus middleware. Due to its novel code transfer mechanism, HOC-SA was extensively evaluated and then adopted as an optional extension to the Globus toolkit as so-called Globus incubator project, available from the Globus site: www.globus.org.

To demonstrate the usefulness of the HOC approach and to provide the reader with the hands-on experience, we describe a broad collection of example applications from various fields of science and engineering, including biology, physics, etc. The Java code for these examples is provided online, complementary to the book. All examples use the HOC Service Architecture (HOC-SA) as their runtime environment. The expected application performance is studied and reported for extensive performance experiments on different testbeds, including grids with worldwide distribution.

The book is targeted at graduate students, software developers, and researchers in both academia and industry. Readers can raise their level of knowledge about methodologies for programming contemporary parallel and distributed systems, and, furthermore, gain practical experience in using distributed software, by picking out topics from the presented material. Practical examples show how the complementary online material can easily be adopted into various new projects.

II Structure of this Book

The book is organized in seven chapters:

Chapter 1 provides an introduction into the area of modern distributed systems and application programming for such systems. We outline the main contemporary approaches to distributed programming and show the benefits and implications of using distributed middleware. In particular, we cover the most popular middleware systems: RMI and CORBA, as well as Web-enabled systems like Tomcat and Globus. We demonstrate that most distributed applications require the advanced functionalities of a software called a “container” like Globus (which provides HTTP data transmission, persistence, security, etc.) and a simple but efficient communication mechanism. This motivates combinations of multiple technologies, e.g., programs that communicate using RMI and are made accessible to clients via a Web service container. Several examples are used to illustrate how such combinations can be set up.

Chapter 2 introduces the main contribution of this book, Higher-Order Components (HOCs). HOCs represent a generic strategy for the implementation of parallel and distributed computation patterns; these patterns accept executable codes and data as parameters which are supplied to them via the network. By enabling type polymorphism for both program and data parameters, HOCs provide clear benefits in the compositional aspects, as shown by the following two case studies. As the introductory case study, a simple but computation-intensive application is used: the calculation of fractal images which are popular in applications of the so-called chaos theory. Two implementations of this case study, both using multiple distributed servers, are shown: one without components and one using HOCs. The comparison of the two solutions demonstrates the benefits of HOCs for the application programmer. The second case study demonstrates the power of code parameters by showing that HOCs can easily be adapted with user-defined code to different computation schemata and thereby fit into a much broader range of applications than are covered by the traditional component technologies.

Chapter 3 is about Web services, a popular Internet-based communication technology, which is used in this book for allowing remote clients to access program components (HOCs). Besides important advantages (e.g., interoperability among software written in different programming languages), some problems of Web service-based systems are discussed and illustrated by an application of the discrete wavelet transform (DWT), in particular: difficulties in transferring executable code and connecting to traditional high-performance computing technology, such as MPI. For resolving such difficulties, this chapter introduces the HOC Service Architecture (HOC-SA) as a runtime environment for HOCs which extends the range of possible parameter types for any Web service. We discuss the management and configuration issues that arise when executable code is used as a Web service parameter and when code is shared among distributed software components. As a new example,

this chapter explains how multiple HOCs can be combined in a single application that adheres to the popular Map-Reduce pattern of parallelism. Finally, we describe the HOC-SA portal – an integrated development environment (IDE) for HOCs – and discuss where the HOC-SA ranges among the other recent developments, such as service-oriented architectures (SOA), the service component architecture (SCA) and the Globus Resource Allocation Manager (WS-GRAM).

Chapter 4 describes applications of HOCs in two different fields, namely, bioinformatics (genome similarity detection) and a physical simulation of deforming objects (e.g., for a car crash test simulation). While the previous examples in this book only operated on synthetically generated input and were used to demonstrate the basic features of HOCs, the applications shown in this chapter are real-world grid applications. The bioinformatics application operates on multiple large genome databases, which could never be fully scanned using a contemporary desktop PC. By the use of a HOC for scanning these databases on the grid with several servers, genome similarities were found which had been previously unknown. The second application allows multiple users to cooperate on the construction of the objects used in the simulation in real time, and thus demonstrates how HOCs can satisfy the requirements of both, multi-user interactive real-time applications and high-performance simulation.

Chapter 5 covers advanced topics, such as the automatic transformation of sequential problem descriptions into parallel code for the execution on top of a distributed platform and the scheduling of component-based, distributed applications on the grid. An analytical cost model for scheduling distributed applications is introduced and experimentally evaluated. An equation system solver is used as an additional case study showing how to apply the previously explained functionalities of software components (i.e., automatic parallelization and scheduling) in concrete applications. It is shown how (and under what circumstances) software development tools from the popular *Apache* Web site, Fractal components and the ProActive grid programming library can be used for creating Web services in grid applications automatically, thus freeing the programmer from dealing with the Web service configuration. Related work, like SOFA components, are also discussed in this chapter.

Chapter 6 ends the book with a concluding discussion of the HOC approach to programming performance-critical applications for distributed platforms and related work. It is shown by examples that the different technologies that recently emerged from Internet-related research areas, namely, the Semantic Web, the Web 2.0 and the grid computing, are complementary to each other: the ontologic classification of data and code in the Semantic Web enables new tactics in algorithms for automatic decision making, which considerably enhance tools like schedulers and code transformation programs (e.g., the ones from the previous chapter); the improved responsiveness of Web 2.0 applications make portal software (such as the HOC-SA portal) much more usable than traditional Web server software, and, thereby, help leveraging grid technologies and tools (e.g., Globus GRAM). An outlook on future work describes planned extensions to the software presented in this book (in

particular, for the HOC-SA, which is available open-source for downloading from the Internet). In particular, extensions to the analytical scheduling model from the previous chapter are discussed, which allow users to predict the performance of distributed computations. Such predictions will help in estimating the runtime of application workflows and the implied resource reservation times, which can be useful for billing and accounting for the usage of network resources.

III Acknowledgments

This book gained a lot from the advice, support and comments from many people.

We appreciate the thorough checking of our manuscript by Herbert Kuchen at the University of Münster, the anonymous referees at Springer-Verlag, and our editor, Ralf Gerstner, at Springer-Verlag.

We owe Cătălin Dumitrescu and his colleague Ioan Raicu at the University of Chicago special thanks for their work on the HOC scheduling experiments on PlanetLab and the DAS-2 Platform. Ian Foster's group in Chicago (especially Borja Sotomayor and Jennifer Schopf) supported us in making HOCs evolve into an Incubator Project in the Globus Toolkit.

Within the scope of the CoreGRID Network of Excellence (funded by the European Commission under the Sixth Framework Programme, Project no. FP6-004265) we had fruitful collaborations and discussions with Jens Müller-Iden, Martin Alt, Christian Lengauer, Eduardo Argollo, Martin Griebel, Michael and Philipp Classen, Marco Aldinucci, Sonia Campa, Marco Danelutto, Murray Cole, Anne Benoit, Nikos Parlavantzas, Françoise Baude, Virginie Legrand, Denis Caromel, and Ludovic Henrio.

And last, but not least, we would like to thank our former Diploma students at the University of Münster, Philipp Lüdeking and Johannes Tomasoni, for their dedicated implementation work on the Alignment-HOC and the LooPo-HOC, respectively.

Jan Dünneweber, 28.01.2009, München Sergei Gorlatch, 28.01.2009, Münster

Higher-Order Components for Grid Programming

Making Grids More Usable

Dünnweber, J.; Gorlatch, S.

2009, XIII, 186 p., Hardcover

ISBN: 978-3-642-00840-5