Preface

Distributed applications are difficult to write. Programmers need to adhere to specific distributed systems programming conventions and frameworks, which makes distributed systems development complex and error prone and ties the resultant application to the distributed system because the application’s code is tangled with the crosscutting concern distribution.

Existing mainstream programming languages, such as Java, do not provide language support for distribution. Rather, programmers must rely on object-oriented distribution frameworks to provide distribution support. Although highly successful, the cost of using these frameworks is that the resultant code is tied to the framework. Object-oriented frameworks in general, and distribution frameworks in particular, can therefore be considered crosscutting in nature because the framework’s code, either via inheritance or containment, is scattered throughout the application’s code thereby tying the application to the framework.

This is a particular concern in distributed systems development because distribution frameworks impose a large code overhead due to the requirements distributed systems impose, such as the need to catch distribution-specific exceptions, locating and binding to distributed objects, locating another server in the event the current server becomes unavailable, and adhering to programming conventions dictated by the framework, such as implementing framework specific interfaces. Consequently, developing distributed applications is complex and error prone and results in application components tied to the distribution framework, which cannot be easily reused outside the application.

In this book we address the above issues and present four contributions to distributed systems development. Firstly, we introduce the concept of a Distribution Definition Language, a high-level domain-specific aspect language that generalises the distribution concern by describing the classes and methods of an existing application to be made remote, the distributed system to use to make them remote and the recovery mechanism to use in the event of a remote error. Secondly, we provide the ability for multiple distribution protocols to be applied to the same code base thereby generalising the distribution concern. Thirdly, we allow the application of distribution awareness to applications in such a way that the application is oblivious of the distribution implementation and recovery mechanism yet is able to fully participate in both. Finally, we provide a simplified approach to the development of distributed systems that allows an application to be either distributed or non-distributed, thereby improving software reuse and simplifying
testability of distributed applications as applications may be functionally tested before having the distribution and recovery concerns applied.

These contributions, by alleviating some of the complexity involved in distributed systems development and by allowing autonomic features, such as recovery, to be transparently added to existing applications, provides a contribution to autonomic computing.

We introduce a software tool in the form of the RemoteJ compiler/generator that uses information contained in the Distribution Definition Language to generate the distributed system specific code and apply it to the application using bytecode manipulation and generation techniques. Finally, we evaluate our contributions and show that the concept of a Distribution Definition Language simplifies the development of distributed applications whilst allowing for greater reuse of application components.
Autonomics Development: A Domain-Specific Aspect Language Approach
Soule, P.
2010, X, 134 p. 39 illus., Softcover
ISBN: 978-3-0346-0539-7
A product of Birkhäuser Basel