JFP **30**, e21, 3 pages, 2020.
doi:10.1017/S0956796820000180

Book Review

Review of "Functional and Reactive Domain Modeling" by Debasish Ghosh, Manning, 2017, ISBN 978-1-617-29224-8

1 Context

Ghosh's book is about the implementation of *Domain-Driven Design* (DDD) using functional programming. This is a timely subject, even though the publication was 3 years ago: DDD is a set of techniques for structuring large software systems and has attained significant popularity, with many books and conferences on the subject. Functional programming also has much to say about structuring software systems, so there are obvious synergies if the two approaches can be made to match. At first glance, this seems unlikely, as DDD comes firmly from object-oriented programming, and the management of state plays a prominent role.

However, this intuition is wrong: Ghosh's book shows that functional programming is a good implementation substrate for DDD. Indeed, functional programming is a better substrate than object-oriented programming: Maintaining a DDD *aggregate*—a unit of transactional integrity—is much easier using immutable data rather than fine-grained imperative operations. In addition, the resulting systems benefit from the programmer's ability to apply algebraic reasoning, use monads to chain processing pipelines, and inject dependencies via free monads for testing.

In addition, the book is an introduction to *reactive programming*, a set of techniques to connect different parts of a system via asynchronous message passing. To do this, the book builds on the functional programming foundations to view reactive programming as a natural extension, using monads and sequences to explain futures and streams.

Thus, the book is fundamentally a book on software architecture. Where many architecture books mainly illustrate architecture in terms of box-and-pointer diagrams, Ghosh presents actual code in Scala, based mostly on the popular *scalaz* and *Akka* frameworks. Ghosh uses a running example—processing bank transactions—to illustrate the implementation of DDD concepts, all the way to a reactive processing pipeline toward the end of the book, along with property-based tests of important properties.

While Ghosh's book is not the only game in town anymore on combining DDD and functional programming, it was certainly one of the first. It is no accident Ghosh comes from an industrial background: DDD is underappreciated in academia, and the functional-programming community would do well to study DDD. This book is a good starting point.

2 Overview

The book starts with a chapter giving an overview of the book's subject: DDD, the advantages of functional programming and immutable data, the need for reactive architecture, and how functional and reactive programming fit together.

Each chapter itself starts with a diagrammatic overview of its contents, and what concrete benefits the sections of the chapter deliver.

In the next two chapters, the book shows how to do domain modeling in the DDD sense using functional programming and Scala: immutable data and static types feature prominently. Ghosh clearly lays out the advantages of the functional techniques, emphasizing how immutability and static types foster composition. The book presents API design in terms of algebraic properties, formulated as types and invariants. The code consistently separates interface from implementation and also introduces smart constructors, lenses, and monads.

The next chapter introduces more algebraic structures, focusing on monoids. Functors, applicatives, and monads are next, with a clear exposition of the roles these abstractions play. These ideas are implemented using the type-class idea from Haskell, using implicits in Scala. Of particular importance in DDD is *validation*, and Ghosh shows how applicatives and monads implement that idea.

The following chapter shows how to modularize code into *bounded contexts*, one of the tenets of DDD, culminating in a presentation of free monads, which decouple interface from implementation.

Subsequently, the book develops the theme of reactive programming. It starts out using the future monad, moving on to reactive streams, and actors. Reactive streams are presented in more detail, as they compose better than actors. Crucially, the book presents *event sourcing*, a technique for structuring the storage of an application not as its current state, but as a log of the things that happened—the events. Event sourcing is another staple of the current DDD scenery (even though it was not part of the original formulation), and a particularly good match for functional programming, as it is inherently based on immutable data.

The book concludes with a chapter on testing using properties with the QuickCheck model, and a summary tying the ideas of the book together one more time.

3 Audience

From the subject, the book has two potential audiences: functional programmers and DDD practitioners, each with an eye toward benefiting from the techniques of the other field. It is a reasonable book for both audiences, but DDD practicioners will get better value. Many of the functional techniques presented—and the clear exposition of their advantages—will be known to functional programmers. Moreover, an in-depth study of DDD still requires a dedicated book on the subject—preferably Evans's original DDD. Some of the presentation is quite specific to Scala and the *scalaz* and *Akka* frameworks, however. Some familiarity with Scala is required.

4 Comparison

Compared to many books on DDD, Ghosh's book is refreshing in two ways: It clearly lays out the advantages of a functional approach to implementing DDD and presents those

https://doi.org/10.1017/S0956796820000180 Published online by Cambridge University Press

advantages using concrete and complete code. Many DDD books—if they present code at all—either only present code sketches or annoyingly imperative code. The obvious point of comparison is Wlaschin's *Domain Modeling Made Functional*, which uses F#. Both are good books, where Ghosh's book more clearly describes the connection to DDD, whereas Wlaschin's book really can serve as an introduction to F#. Thus, the latter has a potentially broader audience.

5 Appraisal of ideas

Looking back, Ghosh's book is an important addition to both the DDD and functional programming literature: None of the ideas presented is individually new, but their combination was at the time of publication. Ghosh convincingly lays out why combining DDD and functional programming is a good idea and describes a complete approach to structuring applications using both. This is especially valuable from an functional-programming perspective, where the literature often gets bogged down in "programming in the small". Moreover, the functional-programming community is quite insular and would do well to connect with the DDD community, which has produced many worthwhile ideas on structuring software. The book is an important first step in that direction.

The book also some weaknesses:

- As mentioned above, the code is quite specific to Scala and the frameworks used and would be significantly different in other languages.
- The book spends quite a bit of space evangelizing the advantages of functional programming, instead of letting the material speak for itself.
- From a functional programming perspective, "functional modeling" as it appears in the title typically means functional techniques for representing domain objects, such as combinator libraries, or algebraic abstractions. However, the actual domain modeling that happens in the book—that of bank accounts, transactions, etc.—is quite traditional. Algebraic abstractions such as monads mostly show up in the representation of processing pipelines, not as an inherent part of modeling itself.

6 Production

The book is quite polished: the writing is lucid, the chapter organization is clear, and the diagrams are well thought-out. Moreover, the book follows a stringent overall organization, where it is always clear where the current material fits in the overall scheme of things.

7 Conclusion

Ghosh's book is an important contribution to the software engineering literature: it convincingly argues for the combination of functional programming and DDD to create software architecture. It is a joy to read and clearly lays out how it is done.

MICHAEL SPERBER
Active Group GmbH
Hechinger Straße 12/1
72072 Tübingen, Germany
E-mail: sperber@deinprogramm.de

https://doi.org/10.1017/S0956796820000180 Published online by Cambridge University Press