Tutorial: Developing Distributed Systems with Multitier Programming

Pascal Weisenburger
Technische Universität Darmstadt
Darmstadt, Hessen, Germany
weisenburger@cs.tu-darmstadt.de

Guido Salvaneschi
Technische Universität Darmstadt
Darmstadt, Hessen, Germany
salvaneschi@cs.tu-darmstadt.de

ABSTRACT

Developing distributed systems is a complex task that requires to program different peers, often using several languages on different platforms, writing communication code and handling data serialization and conversion.

We show how the multitier programming paradigm can alleviate these issues, supporting a development model where all peers in the system can be written in the same language and coexist in the same compilation units, communication code is automatically inserted by the compiler and the language abstracts over data conversion and serialization. We present multitier programming abstractions, discuss their applicability step by step for the development of small applications and discuss larger case studies on distributed stream processing, like Apache Flink and Apache Gearpump.

CCS CONCEPTS

• **Software and its engineering → Distributed programming languages**; Domain specific languages; • **Theory of computation** → *Distributed computing models.*

KEYWORDS

Distributed Programming, Multitier Programming, Reactive Programming, Placement Types, Scala

ACM Reference Format:
Pascal Weisenburger and Guido Salvaneschi. 2019. Tutorial: Developing Distributed Systems with Multitier Programming. In *DEBS '19: The 13th ACM International Conference on Distributed and Event-based Systems (DEBS '19), June 24–28, 2019, Darmstadt, Germany.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3328905.3332465

1 SCALALOCI

In this tutorial, we present the programming style for distributed applications using the multitier programming paradigm [1, 3, 6], which provides mechanisms to abstract over common tedious and error-prone issues of distributed systems development, including network communication, data conversions, and multi-language development.

The multitier approach allows a distributed program to be developed as a single code base generating the code specific to each component of the distributed system – including communication code – automatically during compilation.

In the tutorial, we adopt ScalaLoci [7], a recent multitier language designed as a Scala DSL. ScalaLoci provides *placement types* to associate data and computations to locations. Developers can control the placement by representing the different components of the distributed system at the type level. In contrast to existing multitier languages, ScalaLoci goes beyond the Web domain and the client–server model and enables static reasoning about placement. Also, ScalaLoci supports *multitier reactives* – placed abstractions for reactive programming [2, 4, 5] – which let developers compose data flows spanning over multiple distributed components.

2 TUTORIAL CONTENT

The goal of the tutorial is twofold. On the one hand, we would like to introduce the audience to the general philosophy of multitier programming, showing which fundamental abstractions in this paradigm help programmers to attack the complexity of distributed systems. On the other hand, we would like to concretely guide the audience through the steps of developing an application with ScalaLoci, enabling the attendees of the tutorial to start adopting ScalaLoci for simple projects.

The structure of the tutorial includes three parts.

Multitier programming philosophy and abstractions. In this part we provide an introduction to the design of ScalaLoci. We first cover the general philosophy of developing distributed applications with multitier programming. We discuss the development of multiple peers in the same compilation unit, placement of data and placement of processing functions, and multitier event-oriented communication for coordination. Second, we provide an overview of the main abstractions offered by ScalaLoci and how they can be combined in complex applications. These include placement types, multitier streams, remote blocks, and architecture specifications.

Getting started with multitier programming. This tutorial unit covers the practical details of starting developing applications with ScalaLoci. First, we discuss how to get started with a ScalaLoci project and the generated deployment units. Second we demonstrate how to develop a distributed application by using ScalaLoci step by step. To this end, we use small applications, which are grown incrementally during the presentation, like a minimal online chat, a distributed tweet processing application and a system with a ring topology that passes tokens along the ring.

Multitier programming case studies. The third part of the tutorial discusses how to apply multitier programming to larger applications. Because of the size of these applications, in the tutorial, we only cover the main aspects of refactoring them to multitier programs and the design improvements achieved in the process. We consider two stream processing applications, Apache Flink and Apache Gearpump. In both cases, multitier programming raises the level of abstraction hiding, e.g., communication details, makes the software architecture explicit, and improves safety, replacing a number of potential runtime failures with compile time checks.

Some further topics will be discussed based on time availability and interest of the audience, including the ScalaLoci fault tolerance model and formal models for multitier programs.

3 EXPECTED OUTCOME

The expected outcome of the tutorial is that attendees understand the principles of developing distributed systems with multitier programming and are able to start the development of simple applications based on the ScalaLoci multitier language right away.

4 REQUIRED BACKGROUND

The tutorial does not require any special background. It may fit into a lecture on programming distributed systems in a Master class of a CS course. ScalaLoci is based on Scala, but the tutorial does not require specific Scala knowledge. The Scala syntax will be explained when needed, especially in the aspects that differ from Java. Basic programming skills in a high-level language (e.g., Scala/Java) are necessary to follow the code examples.

5 PRIOR EXPERIENCE

The authors gave a talk on multitier programming for distributed systems in ScalaLoci at the REBLS workshop in 2016 and at the OOPSLA conference in 2018. A demo on the same topic has been given at the <Programming> conference in March 2019.

6 ABOUT THE AUTHORS

Pascal Weisenburger. Pascal is a PhD student at the Technical University of Darmstadt. His research interests focus on programming language design, in particular multitier programming, reactive programming and event-based systems. He is the main developer of the ScalaLoci multitier programming language. His recent publications appear in the OOPSLA conference and in the ECOOP conference.

Guido Salvaneschi. Guido is an assistant professor at the Technical University of Darmstadt. His current research interests focus on programming language design of reactive applications, such as event-based languages, dataflow languages and functional reactive programming. His work includes the integration of different paradigms, incrementality and distribution. He obtained his PhD from Dipartimento di Elettronica e Informazione at Politecnico di Milano, under the supervision of Prof. Carlo Ghezzi with a dissertation on context-oriented programming and language-level techniques for adaptive software. He has co-organized the REBLS workshop at SPLASH for several years and has been program chair of the <Programming> conference. Some of Guido's recent publications appear in OOPSLA, PLDI, ECOOP, ICFP, FSE, ICSE and DEBS.

ACKNOWLEDGMENTS

This work has been supported by the German Research Foundation (DFG) within the Collaborative Research Center (CRC) 1053 MAKI and 1119 CROSSING, by the DFG projects SA 2918/2-1 and SA 2918/3-1, by the Hessian LOEWE initiative within the Software-Factory 4.0 project, by the German Federal Ministry of Education and Research and by the Hessian Ministry of Science and the Arts within CRISP, and by the AWS Cloud Credits for Research program.

REFERENCES

[1] Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. 2007. Links: Web Programming Without Tiers. In *Proceedings of the 5th International Conference on Formal Methods for Components and Objects (FMCO'06).* Springer-Verlag, Berlin, Heidelberg, 266–296. http://dl.acm.org/citation.cfm?id=1777707.1777724

[2] A. Margara and G. Salvaneschi. 2018. On the Semantics of Distributed Reactive Programming: The Cost of Consistency. *IEEE Transactions on Software Engineering, (***TSE***)* 44, 7 (July 2018), 689–711. https://doi.org/10.1109/TSE.2018.2833109

[3] Matthias Neubauer and Peter Thiemann. 2005. From Sequential Programs to Multi-tier Applications by Program Transformation. In *Proceedings of the 32Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '05).* ACM, New York, NY, USA, 221–232. https://doi.org/10.1145/1040305.1040324

[4] Guido Salvaneschi, Patrick Eugster, and Mira Mezini. 2014. Programming with Implicit Flows. *Software, IEEE* 31, 5 (Sept 2014), 52–59. https://doi.org/10.1109/MS.2014.101

[5] G. Salvaneschi, S. Proksch, S. Amann, S. Nadi, and M. Mezini. 2017. On the Positive Effect of Reactive Programming on Software Comprehension: An Empirical Study. *IEEE Transactions on Software Engineering, (***TSE***)* PP, 99 (2017), 1–1. https://doi.org/10.1109/TSE.2017.2655524

[6] Manuel Serrano, Erick Gallesio, and Florian Loitsch. 2006. Hop: A Language for Programming the Web 2.0. In *Companion to the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA*, Peri L. Tarr and William R. Cook (Eds.). ACM, 975–985. https://doi.org/10.1145/1176617.1176756

[7] Pascal Weisenburger, Mirko Köhler, and Guido Salvaneschi. 2018. Distributed System Development with ScalaLoci. *Proceedings of the ACM on Programming Languages* 2, OOPSLA '18', Article 129 (2018).