

Foreword

In the late 1960s, computing practitioners began to realize that the rapid developments in computer hardware—making computers cheaper, faster, smaller, and more reliable—were not being matched by comparable developments in software. On the contrary, it was realized that developing correct, efficient, reliable software was *much* harder than had been generally anticipated. The situation was brought into crisp focus by a stream of high-profile (and highly expensive) software project failures. The term “software crisis” was coined to describe this dismal state of affairs, and thus was born the discipline of software engineering. Describing the software development process as “engineering” made plain the aspirations of the new discipline. The idea was that, ultimately, software development should be an engineering discipline as robust and well understood as other engineering disciplines. Thus, building a large software system should be no more challenging than a civil engineering project such as building a bridge: complex, certainly, but manageable and predictable nonetheless. Of course, things haven’t quite turned out that way, or at least not yet. In the 45 years since the software crisis, we have, for sure, learned a huge amount about the nature of software and software development, and the everyday software applications we take for granted would surely be regarded as miraculous by the early software engineering pioneers. But this undoubted success masks a disappointing truth: software remains hard to develop, and software project failures are far from uncommon. Poorly designed, poorly implemented, and error-prone software is all too common. The discipline of software engineering thus remains as relevant and central to computer science as it was in 1970.

Contemporary software engineering encompasses a wide range of computational paradigms: procedural programming, object-oriented programming, service-oriented programming, aspect-oriented programming, functional programming, logic programming, and so on. Each different paradigm encourages us to think about computation in a different way, and each comes with its own collection of models and development techniques and its own design aesthetic. Logic programming, for example, promotes the idea of computation as automated deduction, while service-oriented computing adopts the idea of programs as service providers. The multi-agent systems research domain, which emerged largely as a subfield of

artificial intelligence in the 1990s, is concerned with building computer systems that can effectively cooperate with each other, and in the 1990s, a number of researchers, myself included, began to think about agents as a software engineering paradigm. If the banner carried by the logic programming community carries the slogan “computation as deduction,” then the banner carried by the multi-agent systems community might read “computation as cooperation,” or perhaps “computation as interaction.” Adopting an agent-oriented view of software engineering implies conceptualizing computer systems as consisting of collections of interacting (semi)autonomous agents: the agents are seen as acting independently in pursuit of goals delegated to them by users. The arguments in support of an agent-oriented software engineering viewpoint are well known, and I won’t rehash them here: for me, the key point is that the most natural way of conceptualizing certain systems is as societies of interacting, semiautonomous agents. Indeed, in a system where control is inherently distributed over multiple stakeholders with potentially competing interests, it is hard to imagine any other reasonable conceptualization. If you accept this, then what follows is the paradigm of agent-oriented software engineering.

If we hope to put agent-oriented software engineering on a par with other software engineering paradigms, then there are a whole raft of issues we need to address. First, and most fundamentally, we need to develop the right conceptual toolkit: What are the key concepts in agent-oriented software engineering that we use in the analysis and design of systems? These concepts will then underpin methodologies for the analysis and design of multi-agent systems, programming languages and development platforms for building and deploying systems, and so on. There have been substantial developments in all of these areas since agent-oriented software engineering was first mooted in the 1990s.

The present volume is a state-of-the-art collection of chapters on agent-oriented software engineering. The chapters presented herein address all the issues that I mentioned above, from methodologies to programming languages and development platforms. While this volume does not mark the end of the story of agent-oriented software development, it does, I think, represent an important milestone in the history of the field and will surely prompt much future research and development.

Preface

Agent-based systems have evolved significantly during the last two decades. The development of such systems involves, among others, artificial intelligence, distributed systems, and software engineering. In this book, we focus on the software engineering facet of agent-based systems, namely, Agent-Oriented Software Engineering (AOSE). In particular, the book consists of a collection of state-of-the-art studies in the AOSE domain. The chapters are organized in five parts: Part I introduces the AOSE domain; Part II refers to the general aspects of AOSE; Part III deals with AOSE methodologies; Part IV addresses agent-oriented programming languages; and finally Part V presents studies related to the implementation of agents and multi-agent systems.

Part I Introduction

This part includes Chaps. 1, 2, and 3 and introduces AOSE as detailed below.

Chapter 1 introduces the notion of software agents with an emphasis on core design and engineering aspects. It elaborates on agent properties and dimensions, emphasizing the novel concepts and abstractions introduced by agent-based systems to software systems' design and implementation.

In Chap. 2, we attempt at defining what AOSE is, make the case for its emergence, and review its evolution throughout the years. We also provide insights into the current status of the AOSE domain and point out future research directions.

Chapter 3, written by Jörg Müller and Klaus Fischer, examines the practical application of multi-agent systems and technologies. The examination is based on a comprehensive survey of MAS deployments and checks the maturity, ownership, application domains, programming languages, and platforms of these deployments. The chapter concludes that MAS applications have been successfully deployed in a significant number of applications and were found to be useful in various market sectors.

Part II Aspects of Agent-Oriented Software Engineering

This part includes Chaps. 4, 5, and 6 and discusses the general aspects of AOSE.

In Chap. 4, we discuss the notion of multi-agent architectures and address the merits of agents and multi-agent systems as a software architecture style.

Chapter 5, written by Joanna Juzziuk, Danny Weyns, and Tom Holvoet, provides a review of the usage of design patterns that are related to MAS. Overall, the authors found that although many patterns exist, these are not well documented, organized, and linked. Thus, the authors provide guidelines for the required efforts in order to increase the usage of such patterns.

Chapter 6, written by Marc-Philippe Huget, overviews the landscape of MAS communication as a major means for applying MAS. In particular, the chapter discusses agent communication languages, ontologies, protocols, dialogue games, argumentation systems, and multiparty communication.

Part III Agent-Oriented Software Engineering Methodologies

This part includes Chaps. 7, 8, 9, and 10 and introduces AOSE methodologies.

In Chap. 7, we discuss agent-oriented methodologies, their desired characteristics, and the extent to which they address these properties. In addition, we review research efforts related to AOSE methodologies.

In Chap. 8, written by Lin Padgham, John Thangarajah, and Michael Winikoff, the authors discuss Prometheus, a well-established and widely used methodology. In particular, they stress the importance of testing within the development of MAS and the challenges that exist in this respect.

In Chap. 9, written by Scott DeLoach, the need for practical, industrial strength of agent-oriented methodologies is emphasized. In this respect, the chapter introduces a customizable methodology that can be adapted and extended for a wide variety of uses.

In Chap. 10, by Jorge Gomez-Sanz, the evolution of INGENIAS (a MAS methodology) is described. In particular, the chapter emphasizes the engineering aspect of designing the methodology and its supporting tools.

Part IV Agent-Oriented Programming Languages

This part includes Chaps. 11, 12, and 13 and presents agent-oriented programming languages.

In Chap. 11, by Mehdi Dastani, a survey of the multi-agent programming research field is presented. In particular, it defines the concepts and abstractions used

in multi-agent systems and the way these are integrated into the agent programming languages and frameworks.

Chapter 12, by Koen Hindriks and Jürgen Dix, introduces a BDI-based MAS programming language that incorporates SE principles. The chapter also demonstrates the use of the language and its success within an exploration game.

Chapter 13, by Olivier Boissier, Rafael Bordini, Jomi Hübner, and Alessandro Ricci, introduces JaCaMo, a platform for multi-agent-oriented programming that incorporates abstractions related to agents, organizations, and environments, which are essential parts of MASs.

Part V Multi-agent Systems Implementation

This part includes Chaps. 14, 15, and 16 and focuses on multi-agent implementation.

In Chap. 14, we survey MAS platforms and frameworks that facilitate MAS implementation. We introduce the reader to a variety of tools and analyze their suitability for MAS implementation needs. The analysis reveals that although many tools were developed over the years, only a few of those are continually being used; it has also become apparent that the evaluations of these tools are rather limited.

Chapter 15, by Renato Levy and Goutam Satapathy, discusses design considerations of very large agent-based systems as applied to an energy distribution use case. They further explore the nuances of the implementation of this use case in CybelePro—an agent infrastructure—and stress the importance of verifying the properties of such systems.

Chapter 16, by Benny Lutati, Inna Gontmakher, Michael Lando, Arnon Netzer, Amnon Meisels, and Alon Grubshtein, introduces a framework for agent-oriented programming for distributed constraint reasoning. The framework facilitates the programming of such agents, the simulation of such systems, and the evaluation of the system's performance.

In this book, we aim to expose the reader to various facets of AOSE. We therefore provide a collection of state-of-the-art studies in this field. We believe that the studies in this book are of interest to researchers, practitioners, and students who are interested in exploring the agent paradigm for developing software systems. We note that although many research efforts have been made in this area, there are many open issues and challenges that need to be addressed and explored.

Beer-Sheva, Israel
Haifa, Israel

Arnon Sturm
Onn Shehory

Agent-Oriented Software Engineering
Reflections on Architectures, Methodologies,
Languages, and Frameworks
Shehory, O.; Sturm, A. (Eds.)
2014, XIV, 331 p. 59 illus., 5 illus. in color., Hardcover
ISBN: 978-3-642-54431-6