Introduction
Alice, Bob, Concurrency, and Distribution

The Importance of Protocols

Two aspects have become pervasive in modern computing: *concurrency*, the performance of multiple tasks at a time; and *distribution*, the usage of components located on different communicating devices. Even mobile phones and tiny single-board computers feature multiple processing units of different kinds, with purposes that go from generic computing to more specialised ones such as artificial intelligence and computer graphics. Computer networks are getting bigger than ever with the rise of the World Wide Web, telecommunications, cloud computing, edge computing, and the Internet of Things. This transformation has caused an explosion in the number of computer programs that communicate with each other over networks – the Internet alone connects billions of devices already.

On one hand, modern computer networks and their applications have become the drivers of our technological advancement. They enable better citizen services, new kinds of industry, new ways to connect socially, and even better health with smart medical devices. On the other hand, these systems and their software are increasingly complex because services depend on other services to function. For example, the website of a national service for citizens might depend on an external identification service to verify that the user can access a certain document. The user's web browser, the website, and the identification service are thus integrated: they communicate with each other to reach the goal of providing authenticated access to documents. In concurrent and distributed systems, the heart of integration is the notion of *protocol*: a document that prescribes the communications that the involved parties should perform in order to reach a goal. We will also refer to communications as interactions.

It is important that protocols are clear and precise. If they are ambiguous, designers of different parts of the same system might interpret the same protocol differently. Different interpretations usually lead to errors and errors can have dire consequences in this setting: applications hanging, data corruption, information leaks, and so forth. The more we equip programmers with solid methods for defining and implementing protocols correctly, the more likely they are to succeed at integrating the different parts of concurrent and distributed systems correctly. The ultimate quest is to increase the intelligibility, reliability, effectiveness, and transparency of these systems, as well as to make people more productive in building them. It is this quest that makes the discipline of interaction worth studying.

Computer scientists and mathematicians might get a familiar feeling when presented with the necessity of achieving both clarity and precision in writing. A computer scientist could point out that we need a good *language* to write protocols. A mathematician could say that we need a good *notation*.

From Protocols to Choreographies

Needham and Schroeder [1978] introduced an interesting notation for writing protocols. A communication of a message M from the participant A to the participant B is written

$$\mathsf{A} \to \mathsf{B} : M.$$

To define a protocol where A sends a message M to B and then B passes the same message to another participant C, we can just compose communications in a sequence:

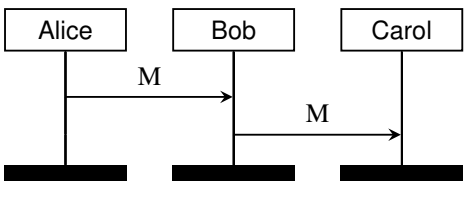
$$\mathsf{A} \to \mathsf{B} : M$$
$$\mathsf{B} \to \mathsf{C} : M.$$

This is called Alice and Bob notation, due to a presentational style found in security research: the participants A and B represent the fictional characters Alice and Bob, respectively, who follow a protocol to perform some task. There might be more participants, like C in our example – typically a shorthand for Carol or Charlie. The first mention of Alice and Bob appeared in the seminal paper by Rivest and colleagues [1978] on their public-key cryptosystem:

'For our scenarios we suppose that A and B (also known as Alice and Bob) are two users of a public-key cryptosystem.'

We ourselves will use fictional characters like Alice and Bob often in this book.

Over the years, researchers and developers created many more protocol notations. Some of these notations are visual rather than textual, like Message Sequence Charts [International Telecommunication Union 1996]. The message sequence chart of our protocol with Alice, Bob, and Charlie looks as follows.



This visual representation (as a message sequence chart) is equivalent to our previous textual representation (in Alice and Bob notation) in the sense that they contain the same information. Intuitively, both notations follow the same style: they define a protocol from the point of view of an external observer, which sees the interactions performed by the participants. In this book, the protocol definitions given in this style are called *choreographies*.

In the beginning of the 2000s, researchers and practitioners started working on languages for writing choreographies that offer more features, for example:

- Including functions for computing the data to be transmitted.
- Nested protocols – that is, the ability to call another protocol as a procedure.
- Manipulating the local memory stores of participants.

We call languages designed for writing choreographies *choreographic languages*.

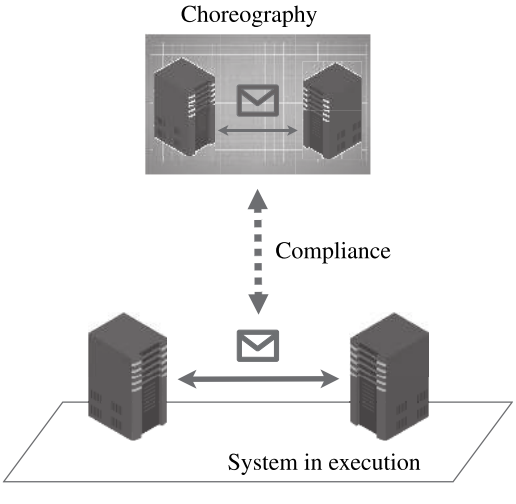


Figure 0.1 An abstract depiction of the property of choreography compliance: the interactions that take place among participants should follow the choreography that has been agreed upon. At the top, the blueprint represents a choreography that prescribes a message communication between two computers (represented by the full arrow with the envelope). At the bottom is the real system where the communication takes place. Compliance (represented by the dashed arrow) guarantees that the real system execution matches the expectations written in the choreography.

Choreographic languages have been adopted in different contexts. In 2005 the World Wide Web Consortium (W3C) – the international standard organisation for the Web – published the Web Services Choreography Description Language (WS-CDL), a language for defining interactions among web services [W3C 2005]. Later, in 2011, the Object Management Group (OMG) – a global consortium for technological standards – introduced choreographies in their notation for business processes (BPMN) [Object Management Group 2011]. The usage of choreographies has been advocated also when dealing with the development of *microservices*, whereby applications are fine-grained compositions of independently executable distributed services [Dragoni et al. 2017]. This momentum has pushed for (and is still pushing for) a lot of research on both the theory of choreographies and its application to programming [Ancona et al. 2016; Hüttel et al. 2016; Giallorenzo et al. 2021]. Alice and Bob are in the spotlight.

Choreography Compliance

Choreographies allow software developers and system designers to formalise an agreement on how the participants of a system should interact. The next step is to develop software and/or hardware that animates each participant according to such agreement. That is, when the implementations of all participants are run together, their joint execution should give rise to the interactions expected by the choreography. When this is the case, we say that the system of participants complies with the choreography, or that the system has the property of *choreography compliance*. (In the literature, compliance is also called *conformance*.)

We depict choreography compliance in Figure 0.1, for a simple distributed system with two participants. At the top we have the choreography agreed upon, depicted as the 'blueprint' that defines the expected communications between the two participants (represented by the computers). At the bottom we have the implementations of the two participants, which are running together and communicating. The property of compliance (represented by the dashed vertical arrow) can then be thought of as the combination of the following two conditions:

1. The system enacts only the communications prescribed by the choreography.
2. Vice versa, the communications prescribed by the choreography are enacted by the system.

Violating the first condition would mean that the system behaves unexpectedly: the system is 'unsound'. Conversely, violating the second condition would mean that the system does not do all that it is supposed to do: the system is 'incomplete'.[1]

Achieving compliance is notoriously challenging. This is not very surprising because coding concurrent systems is hard: programmers have to reason about all the possible ways in which the different participants might interact under all possible schedulings of their actions, which leads quickly to an explosion of the number of cases to be considered [O'Hearn 2018]. The issue is appropriately named the *state explosion problem* in computer science [Clarke & Grumberg 1987; Valmari 1996; Clarke et al. 2011]. It follows that concurrency can look deceptively simple, but in reality even small programs that look innocuous at first sight might yield undesired *emergent behaviour* (the behaviour that emerges from running these programs together). Indeed, programmers do not excel at dealing intuitively with concurrency and distribution, experts included [Lu et al. 2008; Leesatapornwongsa et al. 2016]. Furthermore, achieving choreography compliance is getting more pressing and difficult in practice: with the passing of time, computer networks are getting bigger and including more participants. This trend calls for principles of broad applicability.

The challenge posed by choreography compliance implies that we cannot merely stop at designing precise languages and notations for choreographies. We have to go further and develop rigorous methods for reasoning about the construction of compliant implementations. Motivated by this realisation, researchers have developed several approaches for formally relating choreographies to implementations. As a consequence, choreographic languages are typically designed such that choreographies are mechanically readable, amenable to mathematical reasoning, and used in computer programs [Ancona et al. 2016].

Choreographies in Practice

Some of the methods developed for expressing choreographies and achieving choreography compliance carry principles that can be used on multiple levels. For a programmer, these principles constitute a mental toolbox for the effective development of concurrent and distributed software. The same principles form the backbone of powerful tools, which provide automated or semi-automated help towards the goal of guaranteeing compliance. We now mention some of the most important application strategies for choreographic languages that have been developed so far.

[1] The second condition can be relaxed to allow for systems that do not implement everything prescribed by the choreography while retaining some of the key benefits of the choreographic approach. We discuss this aspect in Chapter 12, after our technical presentation.

Documentation and Specification

The most immediate application of choreographic languages is to reduce ambiguity in the documentation and specification of concurrent and distributed systems [W3C 2005; Object Management Group 2011]. The clarity of choreographies can help with choosing which design best suits the requirements at hand or developing standards for cross-team collaboration.

Furthermore, as we are going to see in this book, some choreographic languages guarantee desirable properties, for example, related to system progress (*liveness* properties). Writing a specification in one such language can therefore be used to prove that the specification respects these properties.

Compilation

Some choreographic languages support the automatic compilation of code for each participant described in a choreography, which then implements correctly what the participant should do [Montesi 2013; Ancona et al. 2016; Autili et al. 2020; Giallorenzo et al. 2021]. This idea has several lines of application.

Scaffolding The generation of skeleton implementations of each participant described in the choreography [Mendling & Hafner 2008; Carbone and Montesi 2013]. Typically, these skeletons are programs with details that need to be filled in, like how the data to be communicated are computed and transformed. Developers are then responsible for manually completing these programs with the missing details. This method has been particularly relevant in the setting of choreographic languages for web services [Object Management Group 2011; W3C 2005].

Libraries The generation of software libraries, which developers can modularly compose and invoke within their applications to make sure that they are following the choreography correctly [Giallorenzo et al. 2020]. For example, a service provider can use this technology to publish a library that clients can adopt to interact correctly with the provided service. While this method might be interesting for any concurrent and distributed system, it is particularly useful for systems that include multiple vendors or implementation technologies, as in cloud computing, edge computing, the Internet of Things, and microservices [Dragoni et al. 2017].

Connectors Choreographies can define protocols for integrating already existing components over a network. These components can be, for example, functions, objects, or services [Carbone and Montesi 2013; Dalla Preda et al. 2017; Scalas et al. 2017; Autili et al. 2020; Giallorenzo et al. 2020]. From the choreography, we can then generate distributed code that steers each component correctly to achieve the desired integration. This application is relevant for different settings, including cloud computing, edge computing, and business processes.

Parallel Algorithms Parallel algorithms, where independent tasks are computed in parallel, can be expressed as choreographies as well [Ng & Yoshida 2015; Cruz-Filipe & Montesi 2016]. In this case, compilation yields distributed software that, when run with the input required by the algorithm, returns the expected result. This application is particularly useful for high-performance computing (HPC) and distributed computation in general.

Verification

Another popular avenue of application for choreographies is the verification of code that already exists. There are two main trends, depending on whether verification takes place before or during execution.

Runtime Verification Given a choreography, we can equip each participant in a system with a monitoring tool which checks that all incoming and outgoing communications comply with what is written in the choreography [Castellani et al. 2016; Neykova et al. 2017].

Static Verification Given a choreography and some existing code for a specific participant, it is possible to automatically analyse the code to check whether it complies with the choreography [Honda et al. 2016; Scalas et al. 2019; Miu et al. 2021]. Choreographic languages can be computationally complete, making the static verification problem undecidable in general. Therefore, static verification typically comes at the cost of weakening the expressivity of the choreographic language or the correctness guarantees provided by choreography compliance.

The applications that we have described require having a choreography, which is usually written manually. When the code of a system is already written, there are methods for the semi-automated or automated reconstruction of a choreography from the programs of participants [Alur et al. 2003; Lange & Tuosto 2012; Lange et al. 2015; Cruz-Filipe et al. 2017; Carbone et al. 2018]. *Choreographic round-trip engineering* is a development process that combines methods for going back and forth between choreographies and participant implementations [Montesi 2013; Carbone et al. 2018]. The former and the latter can be seen as two views that need to be kept in sync. In choreographic round-trip engineering, developers can edit any of the two views and then use (semi-)automated methods to refresh the other. More details and pointers for further reading are given in Chapter 12.

Why This Book

Applications of choreographies rely on a clear understanding of what choreographies are and how choreography compliance can be achieved. Having resources for achieving such an understanding is therefore important, both for revealing how existing tools work under the hood, and for the future development of new technologies and the field of choreography-based development in general. However, at the time of this writing, literature on choreographies consists mainly of research articles that focus on specific developments and are intended for expert readers. There is no well-organised presentation of the key ideas of choreographies aimed at newcomers. This is what motivated the writing of this book, which aims to fill this gap.

More specifically, this book is an introduction to the theory of choreographies and the principles of choreography compliance. We will see how a *semantics* of choreographies can be mathematically defined by using logical methods, which will provide us with an interpretation of what running a protocol means. We will also expose the principles of how choreographies can be correctly implemented in the real world, by defining a translation of choreographies into models of executable programs. Paraphrasing, we are going to study how the Alices and Bobs that participate in a computer system can follow their intended choreographies.

Acknowledgements

I would like to thank the many colleagues – too many to mention here – with whom I have discussed and shared insights on choreographic languages and related topics over the years. The understanding that I gained from these discussions has been invaluable in the process of writing this book.

I am very grateful to the following people for interesting discussions and helpful comments on different contents of this book: Marco Carbone, Ilaria Castellani, Luís Cruz-Filipe, Ornela Dardha, Mariangiola Dezani-Ciancaglini, Simon Fowler, Saverio Giallorenzo, Eva Graversen, Thomas Hildebrandt, Ivan Lanese, Marco Peressotti, Valentino Picotti, Nobuko Yoshida, Gianluigi Zavattaro, and Olaf Zimmermann. Special thanks to Davide Sangiorgi for his advice on the publication of this book.

This book grew out of my teaching experience, which motivated me to explore how different features of choreographies can be presented in a coherent framework. In particular, I would like to extend a special thanks to the students at the University of Southern Denmark who have studied choreographies with me. The experience of interacting with students of different fields (computer science, mathematics, and engineering) played an important role in lowering reading prerequisites and influenced the presentation of several concepts in this book.

I extend my gratitude to the whole team at Cambridge University Press who helped with making the book a reality, in particular David Tranah and Anna Scriven. Thank you also to Johanne Aarup Hansen for her illustration of choreographies, which is part of the cover.

Finally, I would like to thank Maja Dembić for her encouragement and support throughout the process of writing this book.

This Book

Purpose, Audience, and Approach

The aim of this book is to introduce the reader to the theory of choreographies. For newcomers entirely unfamiliar with the idea of choreographies, the goal is to equip them with a fresh perspective on how we can abstract, design, and reason about concurrent and distributed systems. The book explains what choreographies are, how they can be modelled mathematically, and how they can be related to compliant implementations.

The intended primary audience consists of professionals and students in the areas of computer science and engineering, but the book is also designed to be approachable to mathematicians (willing to become) familiar with context-free grammars. Researchers can use the book to acquire the necessary knowledge for advancing the state of the art or for applying choreographies in other contexts. Lecturers should find the book useful in the preparation and execution of courses. Students should be helped in their learning by the rigorous and systematic presentation of the theory. Software architects, developers, and engineers can benefit from the insights in this book to improve their skills regarding integration protocols and the implementation of choreography-based tools. Project leaders can gain a fundamental understanding of the key issues behind systems based on choreographies and how to talk about them.

Pedagogically, the book follows an iterative approach. It starts with a very simple choreographic language and then progressively extends it with more sophisticated features like memory stores and recursion. Each chapter includes examples and exercises aimed at helping with understanding the theory and its relation to practice. Comprehensiveness is not an objective: we will not present features to capture all possible protocols. References to other relevant techniques and further developments are given where appropriate, sometimes in the text but mostly in Chapter 12.

Prerequisites

To read this book, you should be familiar with the basics of:

- The theories of sets, functions, and relations.
- Discrete structures like graphs and trees.
- The induction proof technique, including structural induction.
- Context-free grammars.

It is also assumed that the reader is familiar with the notion of concurrency and the basic intuition of how distributed systems are programmed. Relevant books for covering these topics include [Hopcroft et al. 2003; Tanenbaum & van Steen 2007; Franklin & Daoud 2010; Rosen & Krithivasan 2012; Cormen et al. 2022]. These prerequisites are attainable in most computer science BSc degrees.

To study choreographies, we are going to define choreographic languages and then write choreographies as terms of these languages. The syntax of languages is going to be defined using context-free grammars. To give meaning to choreographies, we are going to extensively use Plotkin's structural approach to operational semantics [Plotkin 2004].

The rules defining the semantics of choreographies are going to be rules of inference, borrowing from logical methods and deductive systems in particular. Knowing formal systems based on rules of inference is an advantage, but not a requirement for reading this book: Chapter 1 provides a brief introduction to the essential knowledge on these systems that we need for the rest of the book. The reader familiar with inference systems (including admissible rules) can safely skip the first chapter and jump straight to Chapter 2.

An important aspect of choreographies is determining how they can be executed correctly in concurrent and distributed systems, in terms of independent programs for *processes*. To model process programs, we will borrow techniques from the area of process calculi. We will introduce the necessary notions on process calculi as we go along, so knowing this area is not a requirement for reading this book. The reader familiar with process calculi will recognise that we borrow many ideas from Milner's seminal *Calculus of Communicating Systems* [Milner 1980].

Structure of the Book

This book is structured in three parts, each one consisting of different chapters:

- Part I introduces a minimal language for defining choreographies and the core theory for relating choreographies to compliant systems of processes.
- Part II extends the choreographic language with standard features from the world of computation: memory, choices, and recursion. This allows for modelling more realistic scenarios.
- Part III presents other extensions and variations of the theory which deal with more specific aspects of some concurrent and distributed systems like asynchronous communication. It also provides references to articles for further reading.

Every chapter contains exercises, which the reader is suggested to solve right where they are presented. For the exercises marked with ↪, a solution is given in the Solutions chapter at the end of this book. The reader is invited to try the exercise first and check the solution later: the solution is provided as a baseline for comparison and as a way to get inspiration in case of getting stuck. Some exercises are marked with !, which indicates that they might be more involved.

Online Resources

A web page containing general information, errata, and additional resources is available. At the time of this writing, it is reachable at

https://fabriziomontesi.com/introduction-to-choreographies/.