Graph Transformation for Software Engineers

Reiko Heckel • Gabriele Taentzer

Graph Transformation for Software Engineers

With Applications to Model-Based Development and Domain-Specific Language Engineering

Springer

Reiko Heckel
University of Leicester
Leicester, UK

Gabriele Taentzer
Philipps-Universität Marburg
Marburg, Germany

ISBN 978-3-030-43915-6 ISBN 978-3-030-43916-3 (eBook)
https://doi.org/10.1007/978-3-030-43916-3

Forewords

Abstraction and modelling are two fundamental conceptual cornerstones of informatics. They are absolutely key to software engineering. Engineers need abstraction to dominate complexity of software design, implementation, deployment and operation. They need precise models to formalise abstractions and reason about them. Graphs are a powerful and general notation that can formally model software structures. They can express snapshots of complex relations among entities of different kinds. Through graph rewriting rules, one can formalise how complex structures evolve over time. By formally analysing models, both statically and dynamically, engineers can verify that the system under design behaves as expected, prior to implementing it and perhaps discovering later that it does not, thus wasting huge investments.

This book fills a much needed gap in the literature. It is the first comprehensive and systematic presentation of graph-based modelling and applications to the practice of software engineering. It can be of use in teaching, to present the foundations of software modelling and verification. It is also a reference book for researchers who are active in software modelling. I have personally often founded my research on graph transformation: in my early years, to formalise data structures and formally analyse them through parsing; more recently, to formalise and analyse spatio-temporal systems (like smart buildings) and their dynamics. I fully share the authors' point that graph transformations are an extremely powerful and tremendously useful tool that can empower software engineers and help them to develop better and higher quality software. This book is a decisive step in this direction.

Carlo Ghezzi

Modern software development is a complex and messy business. Requirements are often incomplete. New development uses complex existing libraries, tools and components that can fail. Multiple development teams proceed on different schedules, making it difficult to assure that their artefacts will "talk to each other" as intended.

I believe that keys to building quality software systems are abstraction and automation. A graph-based approach is a universally applicable and very powerful approach to modelling software at the level of abstraction where its key properties can be represented. Graph rewriting approaches can then support a variety of formal analyses, from requirements completion via property verification to the analysis of product lines. They also support "what if" analysis, allowing developers to determine the impact of their proposed change before investing in fully implementing it. Therefore, it comes as little surprise that graph-based approaches form a basis for many model-driven modelling and development techniques.

I strongly recommend this book to researchers who want to learn about software modelling, and to any senior undergraduate and graduate students who want to be equipped with foundational knowledge and tools to be able to build high-quality, safe software systems.

Marsha Chechik

This book is a pleasant and big surprise. The software engineering community needed such a book, but so far missed experts dedicating their effort to writing it. Graphs and graph transformations play a key role in conceiving, designing, and implementing complex software systems, but they work behind the scenes, so their importance and that of their theoretical foundations is often underestimated. Software engineering is full of models and graphical notations that help experts to tackle very diverse problems, but oftentimes they are defined only informally and lack rigour and precision. Most of these models (can) have a graph-based (abstract) syntax and semantics: this is where the foundations of graphs and graph transformations enter the scene, and this is where a book like this is key.

While methodologies for software architecture design, configuration and version management, and deployment employ diverse graph-based notations, practitioners tend to underestimate the importance of formally defining these models in order to manipulate and transform them in a systematic and sound way. Practitioners and experts in foundations risk ignoring each other; however they represent two sides of the same coin: sound engineering benefits hugely from formal theories, and formal approaches find concrete applications in software engineering. For example, the termination and confluence of a graph transformation system can avoid some tedious problems, while conflicting rules can lead to non-deterministic transformations. These problems can be understood and addressed based on formal foundations. There is a big gap between problems and available theoretical solutions and this book provides an excellent reference guide to help researchers, educators, students, and practitioners to address and solve a large diversity of relevant problems.

Many different software engineering artefacts, including design models, deployment topologies, and development processes, that can be rendered as graphs and manipulated through graph transformations, could benefit from the mature theory developed over the last thirty years. Many solutions have been presented at conferences and workshops, but the necessary coherent collection of their applications to software engineering problems was missing. In the era of systematic literature reviews, this book moves a relevant step ahead, and provides a unique entry point to the main theories and their applications in the context of software engineering, written by two key members of the graph transformation community. Gabi and Reiko did a great job in collecting, harmonising, and presenting all the different findings and solutions in this book. We particularly appreciate the mix of rigour and formality along with proper context and concrete examples.

We are sure that this book will quickly become an essential reference for those interested in the formal underpinnings of graph-based software engineering notations and artefacts, including those interested in exploiting the results presented here to develop original solutions. This book also gives a good overview of the body of work Gabi and Reiko have carried out over recent decades, and we are happy they have decided to present it in this form.

Gregor Engels, Luciano Baresi and Mauro Pezzé

I believe it was in 2010, during the International Conference on Graph Transformation held at the University of Twente, that I expressed the view that the field had matured enough to create both the need to consolidate the state of affairs and the means to fulfil that need. The need, because so many insights had been gained over the years, both about the modelling power of graph transformation and how this can be used in software engineering; insights, however, that were scattered over many papers and in danger of being lost for being insufficiently visible. The means to fulfil the need existed, because the body of knowledge was, by then, sufficiently extensive to merit a monograph where all this could be written up in a unified way.

This is not to suggest that the step of consolidation had not been taken before, successfully. In 1997, the "Handbook of Graph Grammars and Computing by Graph Transformation" appeared, in three volumes, collecting the views and achievements of a large cross-section of the research at the time, doing full justice to its diversity. In 2006, the "Fundamentals of Algebraic Graph Transformation" saw the light, presenting a thorough, unified overview of the theoretical underpinnings, taking into account the (then) most recent advances. Neither of these, however, attempted to convey the message to the software engineering community at large, that solution approaches for some of the pervading problems can be found in graph transformation, while also exposing enough of that field to give newcomers an entrance.

To my delight, I was approached immediately after ICGT 2010 by Gabriele Taentzer and Reiko Heckel, who told me they had concrete plans for just such a monograph as I had imagined. Now these are two scientists thoroughly embedded in the field, who like hardly any others have both an excellent grasp of the theory and an unsurpassed knack for applications in software engineering. In other words, they were the perfect people for the job.

Time has passed since then. For scientists, it is really very hard to find time to write a monograph with sufficient scope, when the day-to-day pressure to also keep contributing to smaller, shorter-term, more urgent activities almost always takes priority. I was very happy to be invited to the project, yet eventually realised that I was unable to find the time to do my bit, and so had to drop out as a co-author. As we all know, however, urgency does not equal importance. I think it is supremely important that a book such as this one has eventually been finished, and I applaud Reiko's and Gabi's perseverance.

The book that lies before you is everything one could wish it to be. Part I presents the necessary background on a sufficiently formal level to be accessible to anyone with a moderate knowledge of discrete mathematics, while at the same time illustrating all presented concepts using recurring, small-scale examples. More importantly still, Part II presents example after example of how all this can indeed be used across the board in all phases of software engineering, from requirements gathering through analysis, design and specification to testing. Not surprisingly, given the close proximity of graphs to (UML-style) models, special attention is paid to concepts of model-driven engineering.

It is a sign of the broad experience of the authors that each and every chapter of Part II is actually based on published research, and ends with extensive pointers to the research literature. Though the book is not meant as a survey, and makes no claims to completeness, it does provide a very good entrance.

Given these many qualities, the potential target audience of the book is diverse. It can be used in academic teaching as the basis for any of a number of courses, complemented with projects to be carried out in any of the topics of Part II; it can act as a great source of reference; but most importantly, it can serve as a means by which researchers and (research-minded) practitioners in software engineering can get to know graph transformation. You can read the book either way: from the more theoretical end, by working your way through Part I and then browsing Part II, or from the more practical side, delving into one of the topics in Part II and where necessary looking up the formal details in Part I. All in all, there is little doubt in my mind that in years to come, this book will be seen to stand out as an authoritative, go-to source of information, indispensable on any (physical or digital) bookshelf.

Arend Rensink

Preface

The digital transformation of society affects all aspects of human life, offering new opportunities but also creating challenges and risks. More tasks will be automated using software. Workflows and business processes are becoming increasingly data driven. Engineering such systems correctly, efficiently and fairly is one of the most critical problems facing us today.

Graphs are of great help when coping with the complexity of software systems. They make explicit the designs of component architectures, process flows and data structures, and provide visual and yet formal representations to analyse them. In order to remain useful and relevant, many real-world software systems are continuously adapted and improved. Their graph-based models need to be transformed to plan or reflect this evolution.

In the Internet of Things, for example, mobile and embedded devices are networked to enable new kinds of applications. We use terms such as "smart home", "smart grid", or "smart city" to describe the highly complex, heterogeneous and adaptive application networks that interact with both human users and their physical environment. The topologies of such networks can be represented by graphs consisting of nodes and edges, where each node represents a device or an application component and each edge models a logical or physical network link. Smart applications are able to adapt their network's topology according to changing needs or context. *Graph transformation systems* are uniquely suited to modelling such adaptations in a direct and visual way. Based on a formal and executable semantics we can validate their operation through simulations and formally analyse their properties.

Today's real-world software systems, built using a variety of languages and technologies and being often distributed, need to be able to evolve in order to remain relevant while allowing integration with other systems. To deal with the resulting heterogeneity and longevity, *model-based software development* lifts essential software engineering tasks to a higher level of abstraction, where we use models to represent the functionality and architecture of applications in a technology-independent, domain-oriented way. This requires concepts and tools that can bridge the gap between models and implementations through code generation, reverse engineering and automated testing.

At the same time, software is becoming more and more data centric, relying on and manipulating structured and unstructured data from a variety of sources. Graphs provide a simple and flexible model for integrating and linking data across formats and applications. They are at the heart of high-level and scalable data representations, such as graph databases designed to store large heterogeneous data sets, using graph-based technologies for efficient querying and transformation.

Model-based engineering employs a variety of modelling languages and techniques targeting application domains, such as Web or mobile applications and cyber-physical or embedded systems. Such *domain-specific modelling languages* require tool support for editing, simulation, compilation, analysis, and version management, which, in order to be produced efficiently and correctly, should be based on definitions of the languages' syntax and semantics. Graphs and graph transformations provide a general mechanism to define and represent models and to specify their manipulation through editing operations, model refactoring, simulation, translation, consistency checking, and synchronisation across languages. They provide a technical (solution) space for domain-specific language engineering, to support the definition and implementation of modelling languages.

Purpose of This Book

Research on graph transformation dates back over 50 years. Yet there is a lack of accessible texts suitable for explaining the most commonly used concepts, notations, techniques and applications without focusing on one particular mathematical representation or implementation approach. To provide a general, widely accessible introduction, the first part of this book will present the foundations of the area in a precise but largely informal way, providing an overview of popular graph transformation concepts, notations and techniques. In the second part, a range of applications of both model-based software engineering and domain-specific language engineering are presented. The variety of applications presented demonstrates how broadly graphs and graph transformations can be used to model, analyse and implement complex systems and languages.

Readers of This Book

We expect this book to be useful and accessible to both current and potential users of graph transformation in the area of software engineering. If you are interested in the use of graph-based modelling and transformation in applications to your own field, this should be the book for you whether you are in academia or industry, had prior exposure to the area or are a complete novice.

While we hope to contribute to the standardisation of notation and presentation, the book is not intended to cover the current state of research. Rather than being comprehensive, we aim to cover work that is both established and stable, inevitably omitting important original results for the sake of presentation.

Although not written as a course book, most parts are suitable for students undertaking postgraduate study at either advanced MSc or PhD level.

Web Site

Further resources, including exercises (with solutions on request) and slides of lectures based on the book, are available at

www.graph-transformation-for-software-engineers.org

with related links and information about updates and corrections.

Acknowledgements

The idea of writing this book was first raised in 2001 by Mauro Pezzé. A first meeting in the afternoon sun during the second workshop on "Graph Transformation and Visual Modelling Techniques" in Crete confirmed our interest in the general concept, but produced little more than an outline before we all went back to our daily business.

Several years went past before the plan was revived by a group including Luciano Baresi, Gregor Engels, Hartmut Ehrig, Mauro Pezzé and the authors of this book, leading to a first collection of draft chapters at various degrees of completion, somewhat heterogeneous in content and style. Unfortunately, activities fizzled out again: probably a case of too many cooks.

When, after several more years, the current authors revisited the concept, Arend Rensink expressed an interest in a book that could provide a comprehensive introduction to graph transformation, and agreed to join the project. With his involvement, we were able to develop most of Part I. Our different backgrounds allowed us to view the same concepts from different points of view and, in fruitful and passionate discussions, to develop a broader, more balanced view of the material now presented in Part I. When, due to changing priorities, Arend was no longer able to contribute, we continued to develop Part II in the same spirit.

We are especially grateful to Ronan Nugent at Springer for his encouragement and exceptional patience. At annual meetings during ETAPS throughout the long years of this project, he provided us with invaluable guidance, helping us to shape and position the book in the best possible way.

The entire book was proofread by Jens Kosiol and, in parts, by Anthony Anjorin, Berthold Hoffmann and Steffen Vaupel, as well as students attending

the module “Formal Methods in Software Engineering” in 2018/19 at the University of Marburg, Germany.

We would like to thank our friends and colleagues who inspired, helped, and motivated us to complete this project. Foremost among them, our friends and mentors Hartmut Ehrig and Michael Löwe are sadly no longer with us. Hartmut was the unstoppable driving force behind the double-pushout approach to graph transformation and a keen promoter of its application to software engineering. Michael was instrumental in restarting the software-engineering-related research during our formative years in Berlin based on his single-pushout approach to graph transformation, which played a big part in directing our own interests.

This book is important to us and, we believe, to the wider graph transformation community in giving access to the wealth of ideas and concepts developed in over 50 years of research on graph transformation and its application to software engineering. As such, it is the first systematic and comprehensive presentation of this range of material directed at an audience beyond the core graph transformation community. This would not have been possible without Arend’s contributions and critical feedback, for which we are especially grateful.

Reiko Heckel and Gabriele Taentzer

Contents